LY21-0571-6

File No. S38-36

IBM System/38

IBM System/38 Control Program Facility Logic Overviews and Component Description

Program Number 5714-SS1

LY21-0571-6

File No. S38-36

IBM System/38

IBM System/38 Control Program Facility Logic Overviews and Component Description

Program Number 5714-SS1

Seventh Edition (September 1985)

This major revision makes obsolete, LY21-0571-5. New components have been added: System/38 Finance Support, Network Facilities, Office Systems, and SNA Distribution Services. Other changes and additions are indicated by a vertical line to the left of the change or revision.

This edition applies to release 7, modification level 0, of the IBM System/38 Control Program Facility (Program Number 5714-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

Use this publication only for the purposes stated in *About This Manual*.

The numbers at the bottom right of illustrations are publishing control numbers and are not part of the technical content of this manual.

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

This publication could contain technical inaccuracies or typographical errors. A form for Reader's Comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department 245, Rochester, Minnesota, 55901. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Contents

About This Manual

The purpose of this publication is to present CPF (control program facility) component level information. When this book is used in conjunction with *What You Should Know*, CPF failures can be isolated to a failing module.

This publication is intended for support level service personnel (program support representatives and technical support representatives). The personnel should complete a customer engineering course in System/38 before attempting to use this document.

The *Introduction* provides general information about CPF. The remaining sections provide specific component information including:

- An overview of the components
- The relationship of the component to other components
- A general description of each of the component modules

Note: This publication follows the convention that *he* means *he* or *she*.

WHAT YOU SHOULD KNOW

To use this publication, you should understand the concepts in the following manuals:

- *IBM System/38 Control Program Facility Concepts Manual*, GC21-7729
- *IBM System/38 Control Program Facility Programmer's Guide*, SC21-7730
- *IBM System/38 Control Language Reference Manual*, SC21-7731
- *IBM System/38 Data Communications Programmer's Guide*, SC21-7825

IF YOU NEED MORE INFORMATION

For more information, refer to the following manuals:

- *IBM System/38 Guide to Publications*, GC21-7726
- *IBM System/38 Guide to Program Product Installation and Device Configuration*, GC21-7775
- *IBM System/38 Operator's Guide*, SC21-7735
- *IBM System/38 Messages Guide: CPF, RPG III, IDU*, SC21-7736
- *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications*, SC21-7806
- *IBM System/38 Diagnostic Aids*, SY21-0584
- *IBM System/38 Functional Concepts Manual*, GA21-9330
- *IBM System/38 Functional Reference Manual*, GA21-9331
- *IBM System/38 Problem Determination Guide*, SC21-7876

SUMMARY OF CHANGES

The following changes have been made to this manual for release 7, modification 0:

- Addition of new components:
  - System/38 Finance Support
  - Network Facilities
  - Office Systems
  - SNA Distribution Services
- Miscellaneous updates and technical changes

This page is intentionally left blank.

Introduction

The CPF (control program facility) is the system support program product for the IBM System/38. The CPF is designed to support the use of interactive work station applications. To supplement the support of the interactive environment, the CPF also provides support for concurrent processing in the batch environment. The CPF is designed to support a wide range of operating environments. No single environment has the exclusive use of a given set of functions. Thus, any user in any operating environment has access to any of the functions.

Some of the functions provided by the CPF are:

- Data base support to rapidly make available, to any job, up-to-date data
- Work management support to schedule quickly and independently the processing of all user requests
- Application development support that allows online development of new applications concurrently with normal production activity
- System operation support that allows the system operator to control the system through the system console or any of the work stations using a control language that provides prompting support for all commands
- Message handling support that allows communication between the system, system operator, work station users, and programs that are executing in the system
- Security support to protect data and other system resources from unauthorized use
- Service support that allows service personnel to diagnose and repair problems or install new functions with minimal impact on normal work flow
- Object management support that allows objects to be grouped and located in the system
- Data management facilities that support both data base files and device files
- Save/restore functions that allow applications and data files to be backed up concurrently with unrelated system operations

The CPF functions are accessed through the use of the control language and the data description specifications. In addition, other program products (such as high-level languages and the interactive data base utilities) also use the CPF functions.

The CPF has many components. These components, processing separately or interactively, provide the support for the CPF functions. Figure 1 shows the CPF components and their identifiers.

The logic diagrams use a heavy line to indicate transfer of control (━━━━━━▶) and a light line to indicate pointers and all other actions (——————→).

The following sections contain descriptions and function overviews for each of the CPF components. The sections are arranged alphabetically by component identifier.

Component	Identifier
Access Path Manager	AP
5250 Information Display System Verification	AR
Binary Synchronous Communications	BS
Command Analyzer	CA
Command Definition	CD
Control Language Compiler	CL
Console Function Manager	CO
Copy	CP
5424 Function Manager	CS
Data Base	DB
Device Configuration	DC
Data Description	DD
Device File Definition	DF
Diskette Function Manager	DK
Common Data Management	DM
3270 Emulation	EM
System/38 Finance Support	FN
Graphics	GD
Installation	IN
Journal Management	JO
Kanji	KJ
Data Base Logging	LG
Librarian	LI
Message Handler	MH
Menu	MN
Network Facilities	NF
Office Systems	OS
5211/3262/3203 Function Manager	PN
Program Resolution Monitor	PR
Prompter	PT
Reclaim/Damage Notification	RC
Service	SC
Advanced Program-to-Program Communications Function Manager	SI
Secondary Logical Unit	SL
Concurrent Service Monitor	SM
Spooling	SP
Save/Restore	SR
Switched Lines	SW
Security	SY
Tape Function Manager	TA
Testing	TE
Commitment Control	TN
SNA-T3	T3
Work Control	WC
Subsystem Description	WD
File Reference Function	WH
Work Station Printer Function Manager	WP
5251 Display Function Manager	WS
Work Monitor	WT
SNA Distribution Services	ZD

Figure 1. CPF Components and Their Identifiers

INTRODUCTION

The access path manager component of the CPF (control program facility) provides a high-level, data stream independent, and device file independent interface to the APPC (advanced program-to-program communications) support provided by the System/38 for devices that operate as an SNA logical unit (type 6.2). The APPC function manager, subsystem monitor, and system arbiter interface with the access path manager to perform all APPC-related operations.

GENERAL OVERVIEW

The APPC function manager, subsystem monitor, and system arbiter create access path control blocks and issue access path manager commands to allocate or deallocate an APPC conversation, and transmit or receive data. The access path manager builds a source/sink request block containing the function request code, option bit settings, transmission data length, and transmission data. Information returned to the APPC function manager, subsystem monitor, or system arbiter includes the received data length, received data type code, error information, and received data, if any.

Access Path Manager Modules

The access path manager component consists of the following modules:

Note: Modules identified with an arrow (-->) are entry modules into the component. Indentation of a module shows its dependency on a previous module.

-->QAPALCON–Allocate Conversation: This module allocates a conversation to an access path control block.

-->QARALSND–Allow Send: This module formats a request that allows the remote program to send application data.

-->QAPCANCL–Cancel Receive Request: This module cancels an outstanding receive request.

-->QAPCRTRB–Create Access Path Manager Request Blocks: This module creates the request blocks used by the access path manager for I/O requests.

-->QAPDLCON–Deallocate Conversation: This module deallocates a conversation from an access path control block.

  QAPDEQUE–Dequeue Outstanding I/O Request: This module waits for the completion of a Request I/O instruction.

  QAPERROR-Access Path Manager Error Handler: This module handles errors resulting from damage to the request block queue.

-->QAPEVOKE–Evoke Program: This module formats a request to initiate a remote program.

  QAPGTSES–Get Session: This module obtains a session for an active conversation.

-->QAPIOCMP–Request I/O Complete Event Handler: This module handles the request I/O complete event.

-->QAPRCV–Receive Input Data: This module requests input data from the remote system.

  QAPDEQUE–Dequeue Outstanding I/O Request: This module waits for the completion of a Request I/O instruction.

-->QAPRSPPS–Send Positive Response: This module sends a positive response to the remote program.

-->QAPSNDER–Send Error Data: This module sends a negative response and error data to a remote program.

-->QAPSNDSG–Send Signal Data: This module sends a signal code (such as a write request) to the remote program.

-->QAPSNDTA–Send Application Data: This module formats a request to send application data to a remote program.

-->QAPUIEH–Unsolicited Input Event Handler: This module handles the unsolicited data events.

-->QAPWAIT–Wait for Input Data: This module waits for input data from the remote system and returns the input data, input data length, data description, and associated indicators.

QAPDEQUE–Dequeue Outstanding I/O Request: This module waits for the completion of a Request I/O instruction.

QAPERROR–Access Path Manager Error Handler: This module handles errors resulting from damage to the request block queue.

-->QAPXMT–Transmit Data: This module transmits all buffered data and requests.

QAPDEQUE–Dequeue Outstanding I/O Request: This module waits for the completion of a Request I/O instruction.

Access Path Manager Operation

Figure AP-1 and the following text describe the operation of the access path manager.

1 When an APPC network is varied on, QLUS (logical unit services process) of the switched line component uses the access path manager to perform the I/O requests associated with negotiating the change number of sessions.

2 When a subsystem is started, the subsystem monitor uses the access path manager to receive program initiation requests from the remote system, if the subsystem description contains a communications entry for an APPC device.

3 A high-level language program, through the QDMCOPEN module of common data management, calls QSIOPEN to open a communications file for I/O processing. QSIOPEN calls QAPALCON to allocate a conversation to the communications device. QAPALCON issues all request I/Os necessary for conversation allocation. QSIOPEN then calls QAPWAIT to wait for the conversation to be allocated. QAPWAIT calls QAPDEQUE to dequeue the allocated conversation request, to set up the access path control block necessary to support the conversation, and to return the session information to QSIOPEN.

4 After the file has been opened, the high-level language program calls QSIPUT to evoke a remote program. QSIPUT calls QAPEVOKE to format the evoke request, and then calls QAPXMT to transmit the evoke request.

5 The access path manager on the remote system receives the evoke request, and returns it to the subsystem monitor, which initiates the requested program. The remote program opens a communications file, allocates its end of the conversation, and calls QSIGET to receive information from the source program.

QSIGET calls QAPRCV to issue the receive request, and then calls QAPWAIT to wait for the completion of the receive. QAPWAIT calls QAPDEQUE to wait for the completion of the receive request I/O.

6 The source program calls QSIPUT to send data to the remote program. QSIPUT calls QAPSNDTA to format the request to send application data.

When one of the transmit buffers becomes full, QSIPUT calls QAPXMT to transmit data to the remote program. QAPXMT uses double buffering during write requests; one buffer is filled while the other is transmitted. When both buffers are full, QAPXMT calls QAPDEQUE to wait for the first transmission to complete processing.

7 When data is received on the remote system, the receive request I/O completes, and QAPDEQUE returns control to QAPWAIT. QAPWAIT returns the data to QSIGET, along with information describing the length of the data received, the type of data, and the current state of the conversation. QSIGET returns the data to the target program.

8 A program may send unexpected data when there is no receive request pending. When this occurs, the APPC station I/O manager sends an event to the access path manager, and QAPUIEH is invoked. QAPUIEH calls the APPC function manager to receive the unsolicited data and process it.

9 The high-level language program may issue a put with invite request. When a put with invite request is issued, QSIGET calls QAPRCV to issue a receive request. When the APPC station I/O manager receives enough data to satisfy the request, an event is sent to the access path manager and QAPIOCMP is invoked. QAPIOCMP calls the APPC function manager to process the completion of the receive request.

10 When one program is done communicating with the other, it calls QSIPUT to detach the other program. QSIPUT calls QAPDTACH to format the detach request. QSIPUT then calls QAPXMT to transmit the detach request. Once the detach request has been transmitted, the programs can no longer communicate with each other.

11 After a communications file has been processed, the high-level language program calls QSICLOSE to close the file. If any data remains in the output buffer at this time, QSICLOSE calls QAPXMT to transmit the data to the remote system. QSICLOSE then calls QAPDLCON to terminate the conversation.

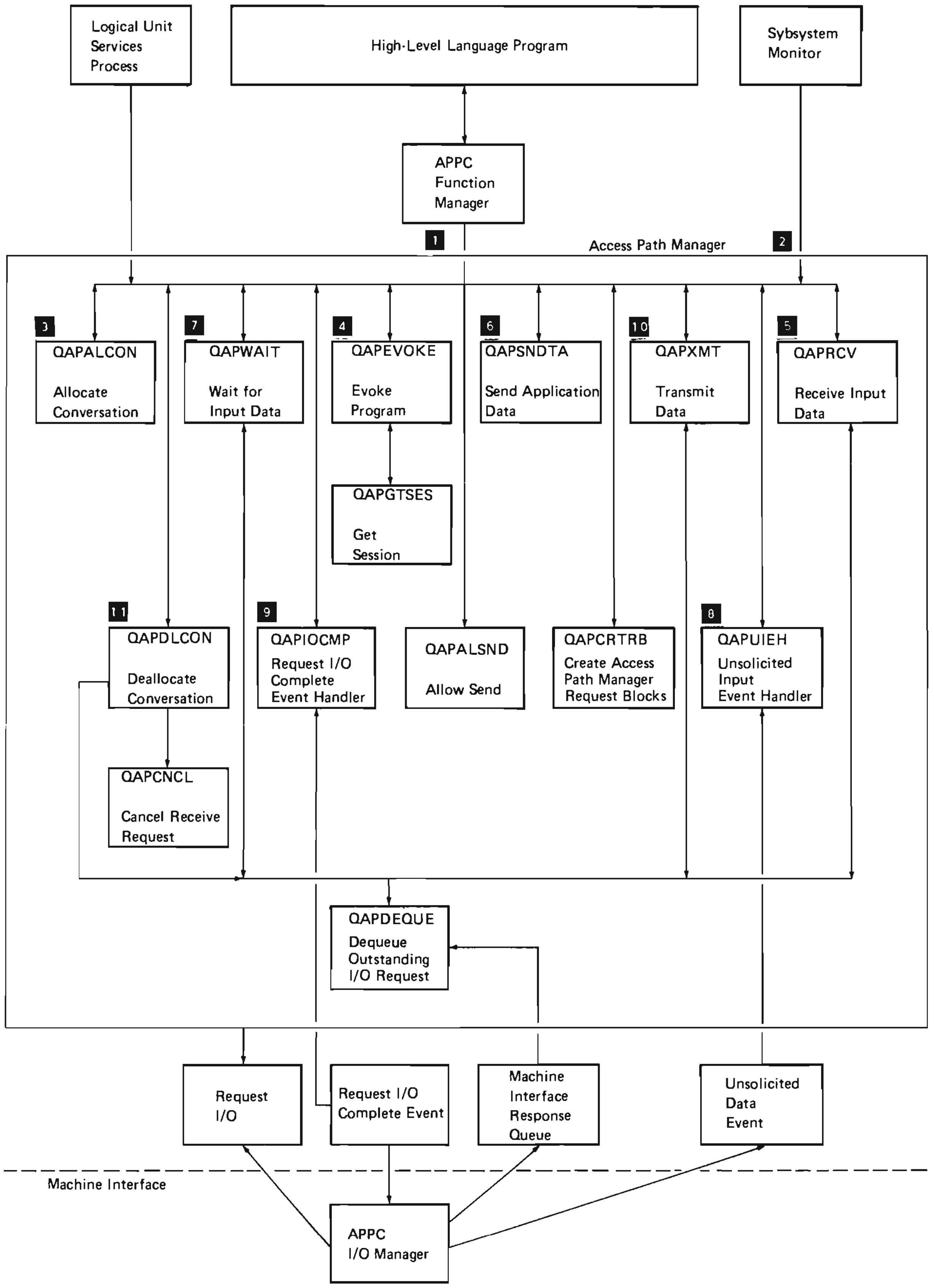


Figure AP-1. Access Path Manager Operation Overview

3270/5250 Information Display System Verification

INTRODUCTION

The 5250 information display system verification component of the CPF (control program facility) provides test request support for the IBM display stations and the IBM work station printers when they are attached to the IBM System/38. The following test request functions are provided:

- Display verification
- Printer verification
- Configuration data
- 5250 ERAP
- Link test

Menus are presented to a work station user to allow selection of these tests. As the selected test is being performed, displays showing test status or additional test instructions are presented to the user. Tests are invoked from the prime option menu, which is presented to the user when the Test Request key is pressed. In addition, the printer verification tests may also be invoked by entering the Verify Printer (VFYPRT) command.

GENERAL OVERVIEW

3270/5250 Information Display System Verification Modules

The modules in this component are divided into six categories:

- Router modules–control the component and determine which test to select.
- Printer verification modules–control verification of the printer.
- Display verification modules–control verification of the display.
- ERAP modules–control error recording analysis procedures.
- Link test modules–control link testing and conversion of console data to 5250 data.
- Configuration modules–display the configuration data of the requesting work station and other work stations and controllers on the work station's line or work station controller.

The 3270/5250 information display system verification component consists of the following modules:

Note: Modules identified by an arrow (-->) are entry modules into the component. Indentation of a module shows its dependency on a preceding module.

Router Modules

-->QARDRIVE–Test Request Driver: This module is loaded when a test request event is processed. It identifies the requesting terminal and opens a device file for that terminal. QARDRIVE then calls QAROPSEL. Upon completion of the requested tests, QARDRIVE closes the device file and terminates the process.

> QAROPSEL–Test Selection Router: This module determines if the requesting terminal is remote or local and if there are any terminals associated with it. It displays the prime option menu so that the user can select a test.

Printer Verification Modules

QARPSEL–Printer Selection: This module builds a list of printers on the same controller as the requesting terminal. This list is displayed to the user so that a printer can be selected to verify.

QARPSTAT–Printer Test Selection and Status: This module sends a menu to the requesting terminal to allow the user to choose how many times to print the test pattern. It also displays completion and error status messages.

QARPRNT–Printer Verification Control: This module controls the printing of the verification pattern.

QAR5256–5256 Print Command: This module executes the print commands common to the 5256 Printer.

QAR5219–5219 Print Command: This module executes the print commands common to all printers and unique to the 5219 Printer.

QAR5225–5224/5225 Print Command: This module executes the print commands unique to the 5224/5225 Printer.

QARKCHAR–Displayable Character: This module displays a chart of all the ideographic characters in the character set for the ideographic display.

QAREVINP–Terminal Input Event Handler: This module monitors input from the terminal while the printer is printing the pattern.

Display Verification Modules

QARDISP–Display Test Selection Router: This module displays the display verification menu to the user so that a function can be selected.

QARATTR–Display Attributes: This module sends the display attributes pattern to the Display. After the user selects an attribute to be displayed, it is displayed on a portion of the display.

QARCHAR–Displayable Character: This module displays a chart of all of the characters in the character set for this device.

QARS2IGC–5553/5224-Model 12 5225-Model 11/5225 Model 12 Print Command: This module executes print commands unique to ideographic printers.

QARSPINP–Specified Input Fields: This module displays several types of field validation to the user. The user can then check for invalid input.

QARFUNKY–Command Function Keys: This module allows the user to test the Roll Up and Roll Down function keys as well as the command function keys.

QARCATTR–Color Display Attributes: This module sends the color display attributes pattern to the display. After the user selects a color attribute to be displayed, it is displayed on a portion of the display.

QARGMENU–Graphics Test Selection Router: This module displays the graphics verification menu so that the user can select a test.

QARGDISP–Display Graphics: This module displays five graphic patterns designed to exercise the graphic capabilities of the display.

QARGVDO–Video Device: This module displays three patterns designed to be used as an aid for calibration of the video device.

QARGPSEL–Graphics Plotter Verification Menu: This module displays the graphics plotter verification menu so that the user may select either option 1 for the 7371 or option 2 for the 7372.

QARG7372–Plotter Graphics: This module produces a test pattern on the IBM 7371 or 7372 corresponding to selection from plotter verification menu.

QARGPRNT–Printer Graphics: This module produces a test pattern on the graphics printer to exercise the graphic printer functions.

ERAP Modules

QARERAP–Device Type Selection for Error Statistics: This module displays the ERAP option menu. It then allows the user to request error history for a type of device (display, printer, controller, or all devices).

QARDEVSL–ERAP Device Selection: This module is used to select a device for which error history information is desired.

QAROUTSL–Output Selection: This module is used to select an output device to which the error information is to be sent.

QARERHST–Error History Table: This module forces the logging of the current data from the controller or work station controller to the system error log. This information (with the information already in the log) and controller (station) statistical data can then be retrieved.

Link Test Modules

QARLINK–Link Test Driver: This module requests the concurrent service monitor to start the link test service function.

QARCONVT–Console to 5250 Data Conversion: This module converts data from a format that can be displayed on the console to a format that can be displayed on an IBM 5250 Display.

QARWSCO–5250 to Console Data Conversion: This module converts the format of data received from an IBM 5250 Display to a format that can be displayed on the console.

Configuration Modules

QARCFIGR–Remote Configuration Data: This module displays information that describes the remote configuration environment of the requesting terminal.

QARCFIGL–Local Configuration Data: This module displays information that describes the local configuration environment of the requesting terminal.

5250 Overview and Relationship to Other Components

Figure AR-1 and the following text describe the operation of the 3270/5250 information display system verification component and its relationship to other components and processes.

1 Pressing the Test Request key on a work station causes a test request event to be signaled.

2 The test request event handler runs in the system arbiter process and notifies the appropriate subsystem monitor process, which then creates a job where QARDRIVE is the problem-state program. This support is provided by the work monitor component.

The six options that can be selected from the prime option menu are:

C	Terminate job
1	Display verification
2	Printer verification (only if a printer is attached to the same controller as the invoking terminal)
3	Configuration data
4	ERAP
5	Link test (only available from a remote work station)

3 The router modules control the operation of the 5250 information display system component and select the proper group of modules to use for the test specified from the prime option menu.

A If option 1 is selected, the display verification modules are used to present the display verification menu. Those modules then process any options selected from that menu and present to the user any displays associated with the options.

B If option 2 is selected, the printer verification modules are used to present the printer selection display so that a printer can be selected for testing. Those modules then process the request and present any displays associated with the test. (This is only available if there is a printer available with the station.)

C If option 3 is selected, the configuration modules are used to present the configuration data displays, either local station or remote station, depending on the type of station that initiated the request.

D If option 4 is selected, the ERAP (error recording analysis procedure) modules are used to present error statistic displays to the user.

E If option 5 is selected, the link test modules are used to invoke the host SDLC link test program. (This is only available from a remote station.)

If option C is selected from the prime option menu, the job is terminated.

4 In addition to option 2 (see 3 B), entry of the Verify Printer (VFYPRT) command uses the printer verification modules to print the test pattern.

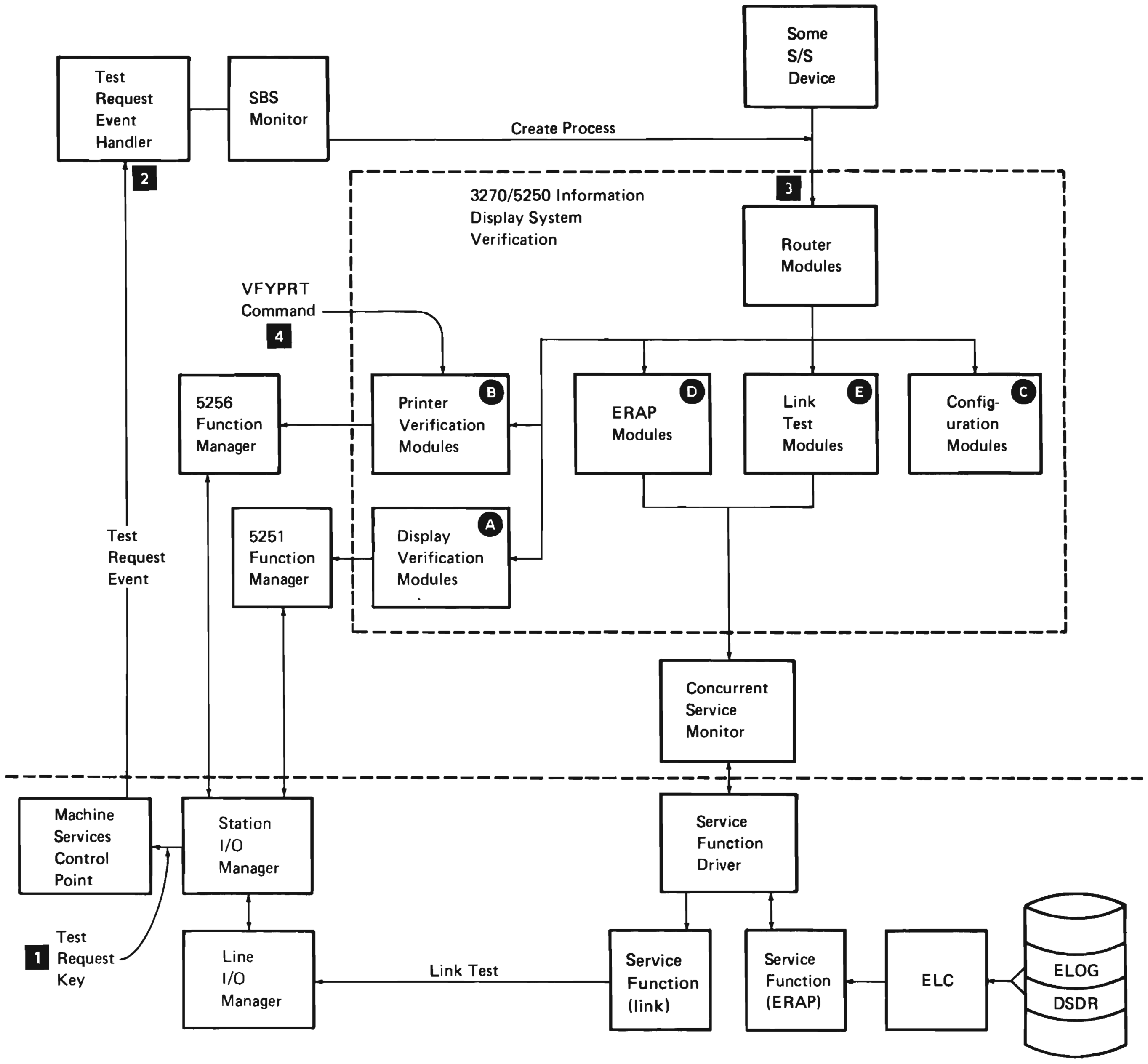


Figure AR-1. 3270/5250 Information Display System Verification Overview and Relationship to Other Components

INTRODUCTION

The BSC (binary synchronous communications) component of CPF (control program facility) provides applications written in RPG III, PL/1, and COBOL a way to transmit data between System/38 and the following IBM systems and devices:

System/Device	Operating With
Series 1	RPS, EDX
System/32	RPG II BSC
System/34	RPG II BSC, ICF BSC
System/3	RPG II Telecommunications Support, BSCA ML/MP Feature, CCP
System/38	RPG III, COBOL
System/370	DOS/VS BTAM, OS/VS1 BTAM/TCAM, OS/VS2 BTAM/TCAM
5280	DE.RPG, COBOL
5110/5120	APL, BASIC
3741 Model 2	
5230 Model 2	
5260	
OS/6	
6670	
5520	
3777, 3776	

The application program uses DDS (data description specifications) to describe the BSC files that are used to communicate between a System/38 and any of the above. Communication is supported on point-to-point switched lines, point-to-point nonswitched lines, and multipoint tributary lines. System/38 performs first level error recovery if there are communication line errors.

The following data management functions are supported by the BSC component:

- Open a BSC file
- Acquire a BSC device
- Put data to a BSC file
- Accept input from a BSC device
- Get data from a BSC file
- Release a BSC device
- Close a BSC file
- Provide error messages and exceptions

GENERAL OVERVIEW

Binary Synchronous Communications Modules

The BSC component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry into the component. Indentation of a module shows its dependency on a previous module.

-->QBSINASP–Initialize LUD-Associated Space: This module initializes the BSC device-dependent portion of the LUD-associated space. It runs under the device configuration component as part of the vary device (VRYDEV *ON) process. This module does not change any lock states, or signal or receive any events. It does not receive or send any messages.

-->QBSOPEN–Function Manager BSC Open: This module completes the path for nonswitched lines between common data management open and the BSC IOM for sending or receiving data over a BSC communication line. It also prepares the path and completes it up to the point of making the actual line connection for switched lines. This module performs the following functions:

- Determines that the device and the ODP (open data path) are available
- Ensures that the device is usable and not in service mode
- Expands the ODP, if necessary
- Initializes the variables in the work area
- Initializes the pointers in the UFCB (user file control block) to the user's buffer
- Calculates and stores, within the function manager work area, addressability to structures and objects required in the function manager
- Performs compatibility checks on various user-specified parameters
- Creates a 23 K object to contain seven request blocks with two source/sink data areas
- Activates the device for a nonswitched line
- Enables the REQIO complete event monitor if the INVITE keyword is defined in the device file

-->QBSPUT–BSC Put: This module is used to pass data, one record at a time, from the user to a buffer. When the buffer is full, it is sent one record at a time, to the BSC IOM for transmission across a line. This module performs the following functions:

- Checks the connection FSM (finite state machine) to determine if a line connection needs to be made and calls QBSFSTIO to make the connection if required
- Calls QBSBID to complete a bid to transmit data
- Obtains the record format for the put and obtains the keywords selected on the put
- Uses the separate indicator area if specified in the device file
- Moves the user's data and appropriate BSC control characters to the source/sink data areas in the request block
- Issues request blocks, one request block at a time, to the BSC IOM
- Utilizes two request blocks (double buffers) to optimize overlapped processing
- Dequeues and reuses request blocks as long as the user continues issuing puts
- Calls QBSPUTCP to issue the last request block and EOT (end-of-transmission) to BSC IOM and sets the connection FSM to contention
- Calls QBSGET, via the INVITE keyword or put or get function, to prepare for receipt of data

-->QBSGET–BSC Get: This module is used to receive data from the BSC IOM. It then passes the data, one record at a time, to the user. This module performs the following functions:

- Checks the connection FSM to determine if a line connection needs to be made and calls QBSFSTIO to make the connection if required
- Calls QBSBID to complete a bid to receive data
- Sends receive request I/Os to the BSC IOM
- Utilizes two request blocks (double buffers) to optimize overlapped processing
- Dequeues the next request block when data from a previous request block has been exhausted
- Finds the record format that it is to use to process the received data, via record ID processing, a default format, or user-defined format
- Sets response indicators as specified by the record format
- Uses the separate indicator area if specified in the device file
- Moves the received data to the user's input buffer and updates the I/O feedback area
- Receives an EOT and sets the connection FSM to contention

-->QBSFSTIO–First I/O: This module is used to interface via the CPF LUD-opened event with the switched lines component to establish a switched line connection. The connection is made on the first put or get operation after an open.

-->QBSBID–Bid for Line: This module sets the line connection of the FSM to put or get. All outstanding put and get request blocks are dequeued. It also issues a request I/O for a bid and dequeues the bid feedback record. If an error occurs in the feedback record, this module calls QBSERP.

-->QBSPUTCP–Put Complete: This module is used to issue the last request block (with proper ending and EOT) and to dequeue all outstanding put request blocks.

-->QBSCLOSE–Function Manager BSC Close: This module is used to complete the I/O if required, by calling QBSPUTCP, and to clean up the system objects so that the BSC device can be reused. It closes a BSC file or releases a BSC device.

-->QBSERP–Error Handler: This module is used for all error handling within the BSC component. QBSERP provides all of the error messages and exceptions between the user or user programs for the BSC component. When necessary, it does a request I/O (continue) for the I/O modules. It also handles the setting of certain areas in the ODP, function manager work areas and LUD-associated space when appropriate. These areas are:
- Device unusable flag
- File in error-set by QDMERRHP, which is called by QBSERP
- Major-minor code for exceptions
- State of BSC FSM

-->QBSASYNC–BSC Asynchronous Input Request I/O Handler: This module receives control when a bid or data request I/O event completes. After dequeing the request block, QBSASYNC signals the data available event. Data management calls QBSGET to process the data.

Binary Synchronous Communications Overview

Figure BS-1 and the following text describe the operation of the BSC function manager.

1 QBSINASP is called by the device configuration component and is part of the vary on process. QBSINASP initializes the device-dependent portion of the LUD-associated space and passes control back to the device configuration component.

2 A high-level language program, through the QDMCOPEN module of common data management, calls QBSOPEN to complete the opening of a BSC or mixed device file.

QBSOPEN is also called by common data management to perform subsequent acquires of a program device.

A An argument list is passed that contains a pointer to the UFCB (user file control block) and an index into the device name list for the device-dependent open.

B The message handler is called to signal exceptions to the user.

3 After the file has been opened, information can be passed by the user to the BIOM (BSC I/O manager) or to the MIOM (MRJE I/O manager) by calling QBSPUT.

A An argument list is passed that contains a pointer to the UFCB, a pointer to the option list, and a pointer to the control list. The option list is not used by the BSC component. The control list indicates which record format in the device file should be used for this request.

QBSPUT checks the connection FSM to determine if a line connection needs to be made. If a connection does not exist, QBSPUT calls QBSFSTIO to establish a connection. QBSFSTIO sets the connection FSM to a contention state. QBSPUT then calls QBSBID to complete a bid to transmit data.

When one of the put buffers becomes full, or a special function is requested through the use of communications file keywords, QBSPUT issues a request I/O to the BIOM or to the MIOM to send data to the remote station. QBSPUT calls QBSPUTCP to issue the final request I/O of the session when the application requests via a file keyword.

QBSPUT uses double buffering during put requests; one buffer is being filled while the other is being transmitted.

B The message handler is called to signal exceptions to the user.

4 After the file has been opened, information can be received from the BIOM or from the MIOM by calling QBSGET.

A An argument list is passed that contains a pointer to the UFCB, a pointer to the option list, and a pointer to the control list. The option list is not used by the BSC component. The control list indicates which record formats in the device file should be used for this request.

QBSGET checks the connection FSM to determine if a line connection needs to be made. If a connection does not exist, QBSGET calls QBSFSTIO to establish a connection. QBSFSTIO sets the connection FSM to a contention state. QBSGET then calls QBSBID to complete a bid to receive data. When the bid is complete, QBSGET sends two receive request I/Os to the BIOM or to the MIOM. QBSGET checks for any input data not already deblocked from a previously received request I/O before dequeuing the next RB. QBSGET receives the data and passes it to the user's input buffer.

QBSGET calls QBSPUTCP when the user issues a get following a put, and the FSM indicates a put state.

QBSGET uses double buffering during get requests; one buffer is being emptied while the other is being received.

B The message handler is called to signal exceptions to the user.

5 After a BSC file has been processed, QDMCLOSE calls QBSCLOSE to close the file.

A An argument list is passed that contains a pointer to the ODP control block, an index into the ODP device name list, and the type of close to perform. A temporary close is invalid for BSC. If a temporary close is encountered, an exception will be signaled.

If QBSCLOSE is processing a permanent close or a nonreclaim TCLOSE (normal) and the file-in-error bit is not set and the FSM indicates put state, QBSCLOSE calls QBSPUTCP to transmit any data remaining in the buffers and sends a normal EOT.

B The message handler is called to signal exceptions to the user.

6 When a put with invite request is specified, and the INVITE keyword is defined in the device file, the completion of a request I/O causes QBSASYNC to execute. QBSASYNC dequeues a request block and signals the data available event. If an accept input to the BSC device is issued, data management calls QBSGET to process the records in the block.

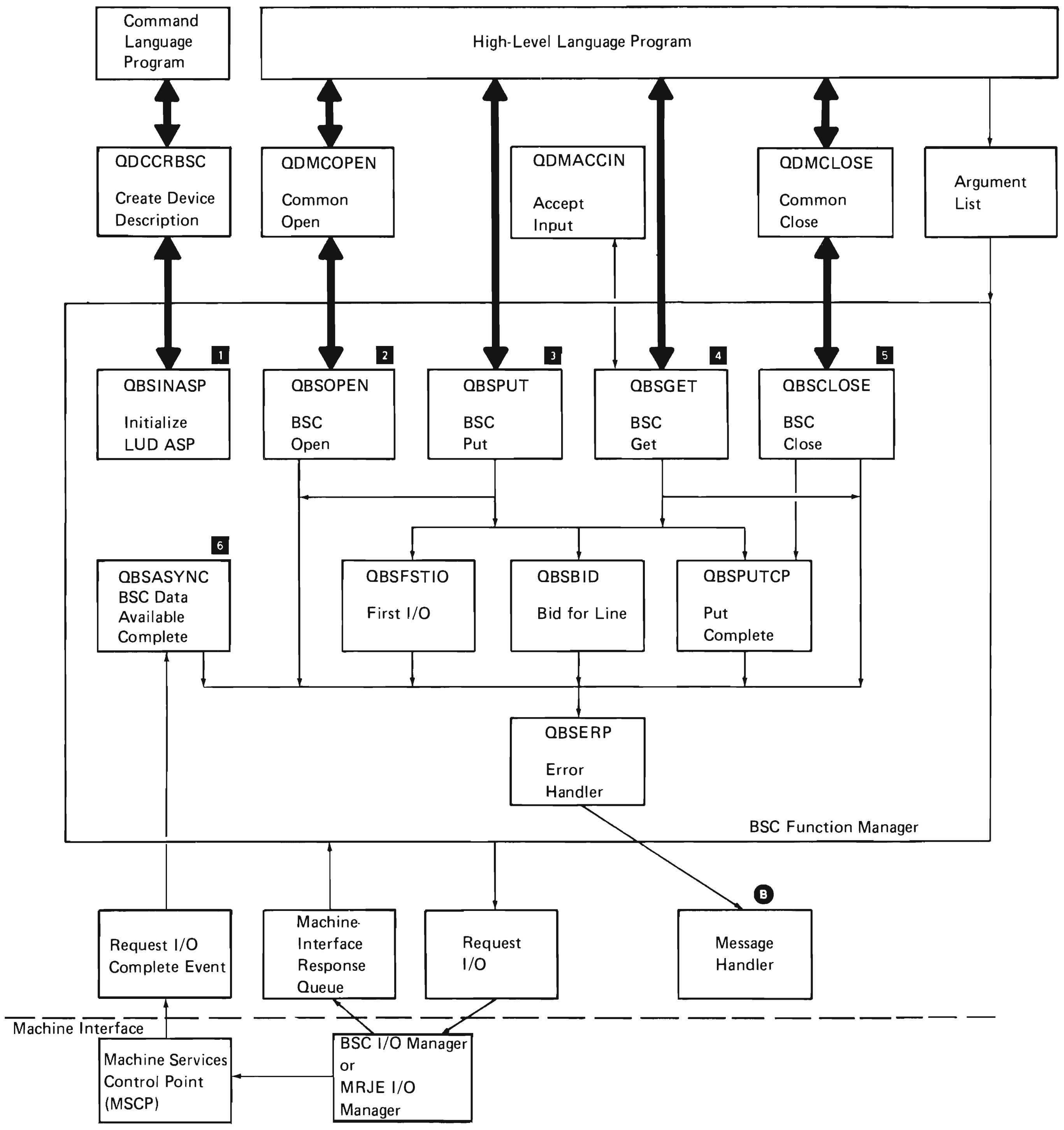


Figure BS-1. BSC Function Manager Operation Overview

INTRODUCTION

The command analyzer component of the CPF (control program facility) processes all commands. Command processing consists of parsing the command, validity checking, transferring control to a command processing program or to an application program, and returning to the caller of command analyzer.

The command analyzer uses information contained in a CDO (command definition object), which is created by the command definition component, to perform validity checking and to determine the format of parameters to be passed to application programs or command processing programs. The CDO contains a description of the command, the name of any user-defined validity check program, and the name of the command processing programs that processes the command. Each command has its own unique CDO.

Validity Checking

Validity checking ensures that the required parameters for a command are entered and that any values specified are allowable values. If a command does not meet its validity checking requirements, a message is sent to the user describing the errors. The command can then be corrected and reentered.

Validity checking standards for a single keyword allow for:

- Values to be restricted to a list or range of values
- Transformation (mapping) of an input value to another value
- Values that meet the syntax requirements for NAME, DATE, TIME, NUMERIC, GENERIC-NAME, and so forth

Validity checking can be used to test the relationship of multiple keywords by:

- The existence of a keyword
- Comparing a keyword value to
  - Another keyword
  - A constant

The user can also define validity checking programs to supplement the validity checking of the command analyzer.

Parsing a Command

Parsing a command consists of taking the command and converting it into a format that can be used by the application program or command processing program.

The command analyzer receives a work area from the caller that contains the length of the command string and the command string itself. The command string is processed and a token list is created from it.

Length of Command	Command String	End of Cmd ID

Example of Work Area

The token list is a list of elements, with each element containing information about a part of the command (command name, keyword, keyword value, qualified name, list, number of list elements, and length). A token list element consists of three parts:

- Attribute byte: defines the command part (token) being described
- Length: defines the length of the keyword value or list of keyword values
- Value of the keyword or number of list elements if the attribute byte defines a list of keyword values

The token list is then processed and, using information contained in the CDO, a positional list is created.

Attribute	Length	Value or Number of List Elements

The positional list contains elements that further describe the keyword values. Each element contains the following information:

- Four bytes of attribute information
- A link to the next element
- A displacement value into the CDO that locates the information about the keyword value
- A displacement value into the work area where the converted value will be placed
- The length of the value data or the displacement to the first element if the value is a list of values
- The value data itself or the number of elements if the value is a list of values

Attribute	0	CDO Displacement	Work Area Displacement	Length of Data or Displacement to First Element	Data or Number of Elements

Link to next element (0 when last element for a parameter)

If the command is to be processed, an argument list containing a pointer to each of the converted data values in the work area is passed to the application program or command processing program. If the command is to be processed at a later time, the positional list is passed back to the caller.

Figure CA-1 shows an overview of command processing.

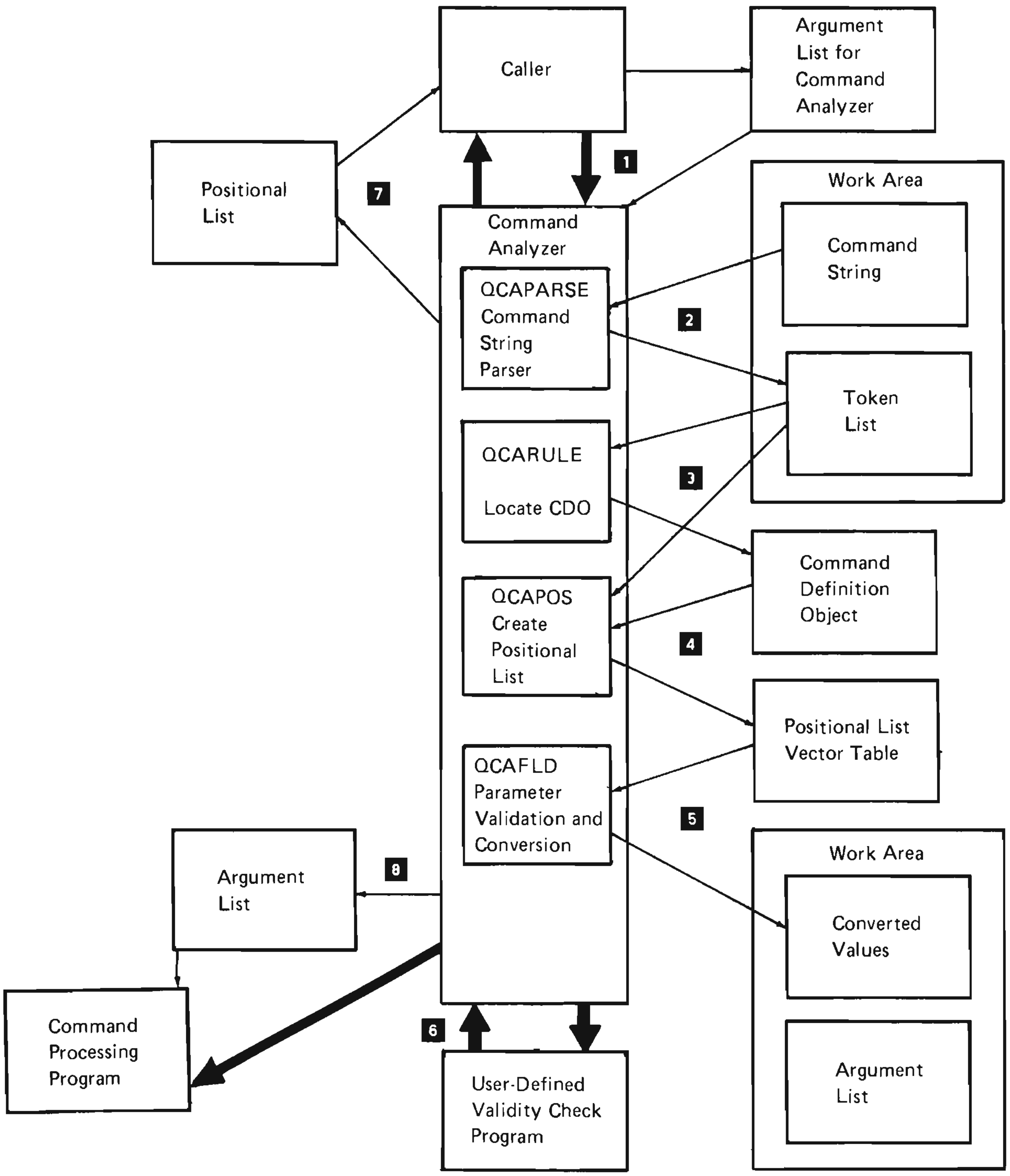

Caller
Argument List for Command Analyzer
1
Positional List
7
Command Analyzer
Work Area
Command String
QCAPARSE Command String Parser
2
Token List
QCARULE Locate CDO
3
Command Definition Object
QCAPOS Create Positional List
4
Positional List Vector Table
QCAFLD Parameter Validation and Conversion
5
Work Area
Converted Values
Argument List
Argument List
8
Command Processing Program
6
User-Defined Validity Check Program

Figure CA-1. Command Processing Overview

Command Processing

1 The command analyzer is called, with a pointer to the positional list passed as an argument. Within the positional list is a pointer to a work area. The work area contains the command string.

2 QCAPARSE locates the command in the work area and creates a token list from the various parameters of the command. The command parameters can be entered in both positional and keyword formats on a single command (each parameter can be specified only once). The parameter values can be simple scalars, qualified names, or lists of values (not necessarily in the order required by the CDO). If a parameter is entered in keyword format, the parameter value is associated with the keyword name for use by QCAPOS.

3 QCARULE finds the command name in the token list and locates the CDO for this command. It also validates the command environment and mode.

4 QCAPOS reads the token list and creates a positional list that will contain the keyword values and defaults as specified by the CDO. Defaults are provided by the CDO for unspecified optional parameters. A vector table containing displacement values is built into the positional list to provide ordering of the parameters as specified in the CDO.

5 QCAFLD validity checks the parameter values (scalar value checks, interparameter checks, and user-defined validity checks) as specified in the CDO. The converted data values are then placed in the work area.

6 If specified in the CDO, the user-defined validity check program is called to perform extended validity checking.

7 If the command is *not* to be executed immediately, the command analyzer returns to the caller with the positional list.

8 If the command is to be executed immediately, the command analyzer transfers control to the proper command processing program passing an argument list that contains pointers to the converted data values.

GENERAL OVERVIEW

Some CPF components use the command analyzer only to parse and validity check commands. The parsed and validity checked form of the command is placed in a positional list. The positional list is then passed back to the caller to be used when the command is executed at a later time. Components that use the command analyzer only to parse and validity check commands are:

- CL compiler (as the commands are being compiled)
- Command definition
- Spooling (for job commands)
- Prompter (for partial commands as they are entered from the console or work station)

Some system functions and programs use the command analyzer to parse, validity check, and immediately execute commands. These are:

- Interpretive CL processor
- CL programs (to validity check and then execute the commands)
- Prompter (to execute a command after it is completely entered and processed)
- High-level language programs (to process commands using the QCAEXEC interface); (see *Command Analyzer Modules* for a description of the QCAEXEC module functions.)
- Source entry utility (for validity checking only)

Two interfaces are provided to the command analyzer. The QCAEXEC interface is used by all high-level language programs. The parameters passed to QCAEXEC, as seen from a high-level language program, are:

- A command string to be processed
- A packed (15, 5) numeric value that specifies the length of the command string

QCADRV is the interface for the other CPF components.

Command Analyzer Modules

The command analyzer component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QCADRV–Command Analyzer Driver: This module provides the interface between other CPF components and the command analyzer. It controls the module flow during the command analysis process.

- QCAPARSE–Command String Parser: This module scans the command string and creates a token list containing the keyword values.
    - QCAPRBLD–Parser Table Build Routine: This module builds the parser tables needed by QCAPARSE.
- QCARULE–Locate Command Definition Object: This module establishes addressability to the CDO and validates the command environment and command mode.
- QCAPOS–Create Positional List: This module creates a positional form (positional list) of the command and adds defaults, as specified in the CDO, if necessary.
    - QCAFSCAN–Scan Character Variable: This module scans the contents of character variables used on CL commands, classifies its token type, and builds a token list element.
- QCAFLD–Parameter Validation and Conversion: This module uses information in the CDO to convert and validate individual field data.
    - QCAFEXPR–Process Expression: This module processes and validates control language expressions.
    - QCAFBIF–Process Built-in Function: This module processes and validates built-in functions.
        - QCABIFV–Built-in Function Validity Checker: This module validity checks the number and value of built-in function arguments.
    - QCAFCMD–Process-Embedded Commands: This module processes embedded commands on TYPE (*CMD) parameters.
- QCAIFLD–Perform Interparameter Checks: This module performs validity checking used to test the relationship between keywords.
- QCACALL–Interpretive Call Processing: This module invokes an application program when a call is encountered.
- QCATRS–Create Argument List and Invoke Command Processing Program: This module invokes user-defined validity check programs and command processing programs.

-->QCAEXEC–High-Level Language Interface to Command Execution: This module provides the interface between a high-level language program and the command analyzer. It converts commands in a high-level language program into a form that can be used by the QCADRV interface. QCAEXEC moves the command and command length into the work area and passes this information to QCADRV. QCADRV then processes the command.

The following module is used by most of the command analyzer modules:

QCAXTND–Extend Command Analyzer Space Objects: This module extends the space of a command analyzer positional list or work area if additional space is needed.

Command Analyzer Overview

Figure CA-2 shows the components and functions that use the command analyzer and the components and functions used by the command analyzer to perform its tasks. Following Figure CA-2 are other figures that show specific component relationships to provide command analyzer functions; with each figure is a description of the relationship.

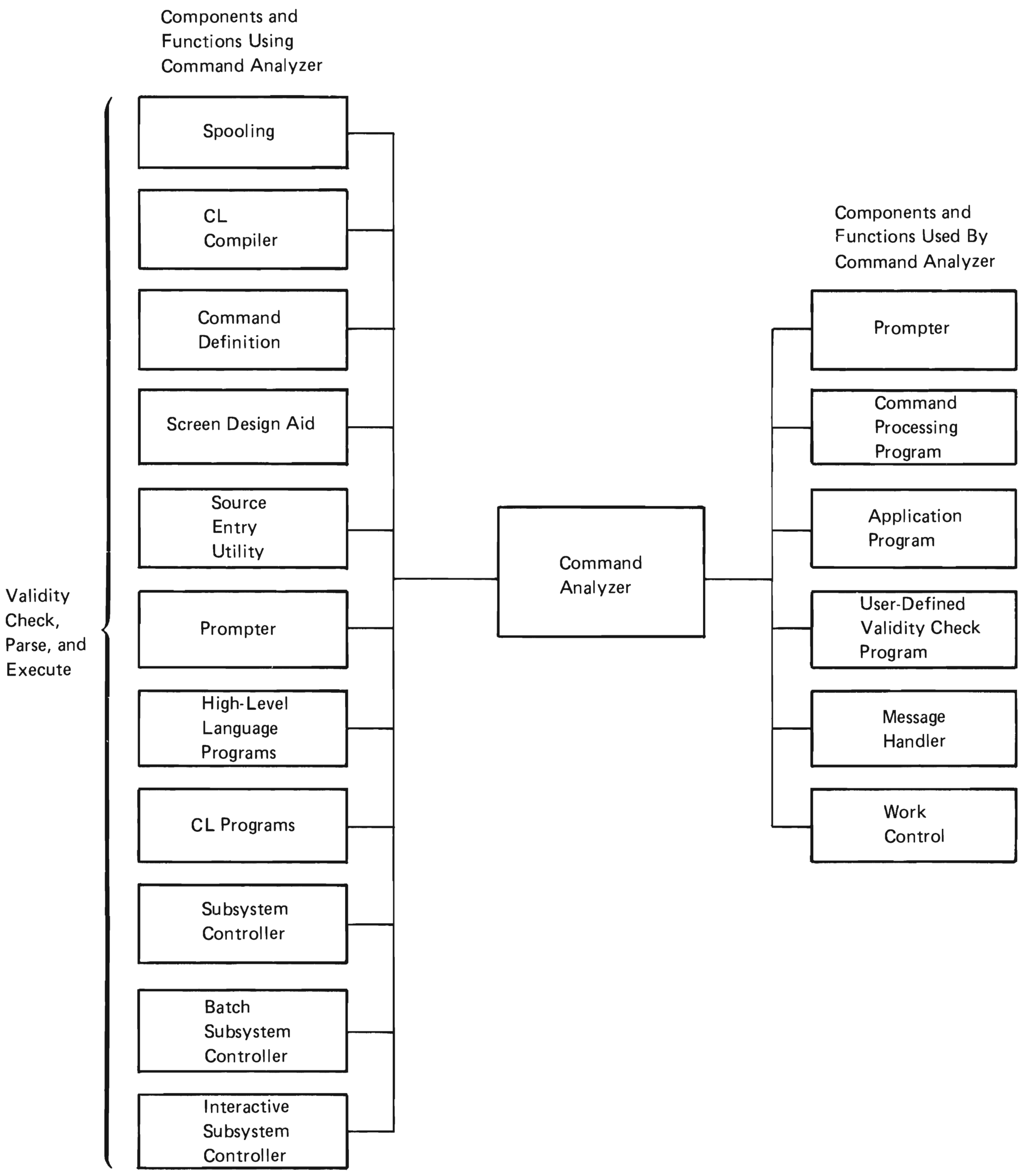

Components and Functions Using Command Analyzer
Validity Check, Parse, and Execute
Spooling
CL Compiler
Command Definition
Screen Design Aid
Source Entry Utility
Prompter
High-Level Language Programs
CL Programs
Subsystem Controller
Batch Subsystem Controller
Interactive Subsystem Controller
Command Analyzer
Components and Functions Used By Command Analyzer
Prompter
Command Processing Program
Application Program
User-Defined Validity Check Program
Message Handler
Work Control

Figure CA-2. Command Analyzer Overview

Command Analyzer as Used by Spooling, CL Compiler, Prompter, Command Definition, and the Source Entry Utility

1 A command is read by one of the components shown in Figure CA-3, and the command analyzer is invoked to parse and validity check the command. The component using command analyzer moves the command string and its length to the work area. An argument list containing a pointer to the positional list space is passed. The positional list contains a pointer to the work area, the option bytes, and the return value. The work area contains the command string and the length of the command string.

2 Command analyzer, using information in the CDO, validity checks the command.

3 If specified in the CDO, the user-defined validity check program is called to perform extended validity checking.

4 Control is returned to the caller. A positional list and work area containing the parsed command and assigned default values are passed back at the completion of parsing and validity checking.

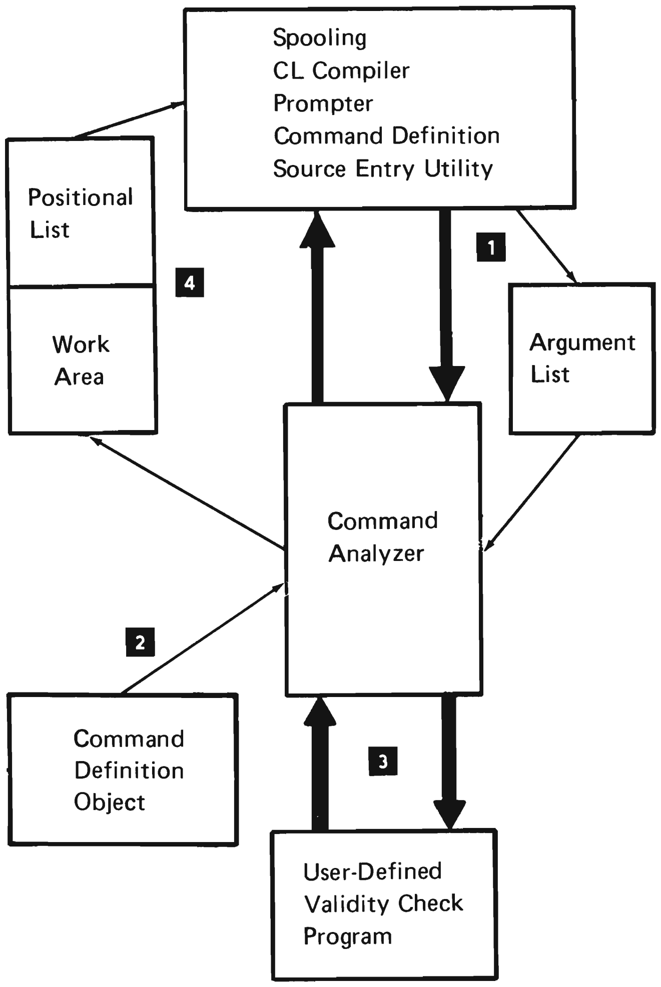


Figure CA-3. Command Analyzer as Used by Spooling, CL Compiler, Prompter, Command Definition, and the Source Entry Utility

Command Analyzer as Used by a CL Program

Figure CA-4 and the following text describe how the command analyzer is used by a CL program.

1 The CL program assigns values to the symbolic variables and invokes the command analyzer. A token list, which resides in the associated space of the CL program, is passed.

2 The command analyzer, using information in the CDO, validity checks the parameters and performs interparameter checks in the command.

3 If specified in the CDO, the user-defined validity check program is called to perform extended validity checking.

4 The command analyzer transfers control to the proper command processing program. An argument list containing pointers to the converted data values in the work area is passed.

5 If an error is detected during the command analysis process, an exception is signaled and control is passed back to the CL program.

6 The command processing program returns control to the CL program.

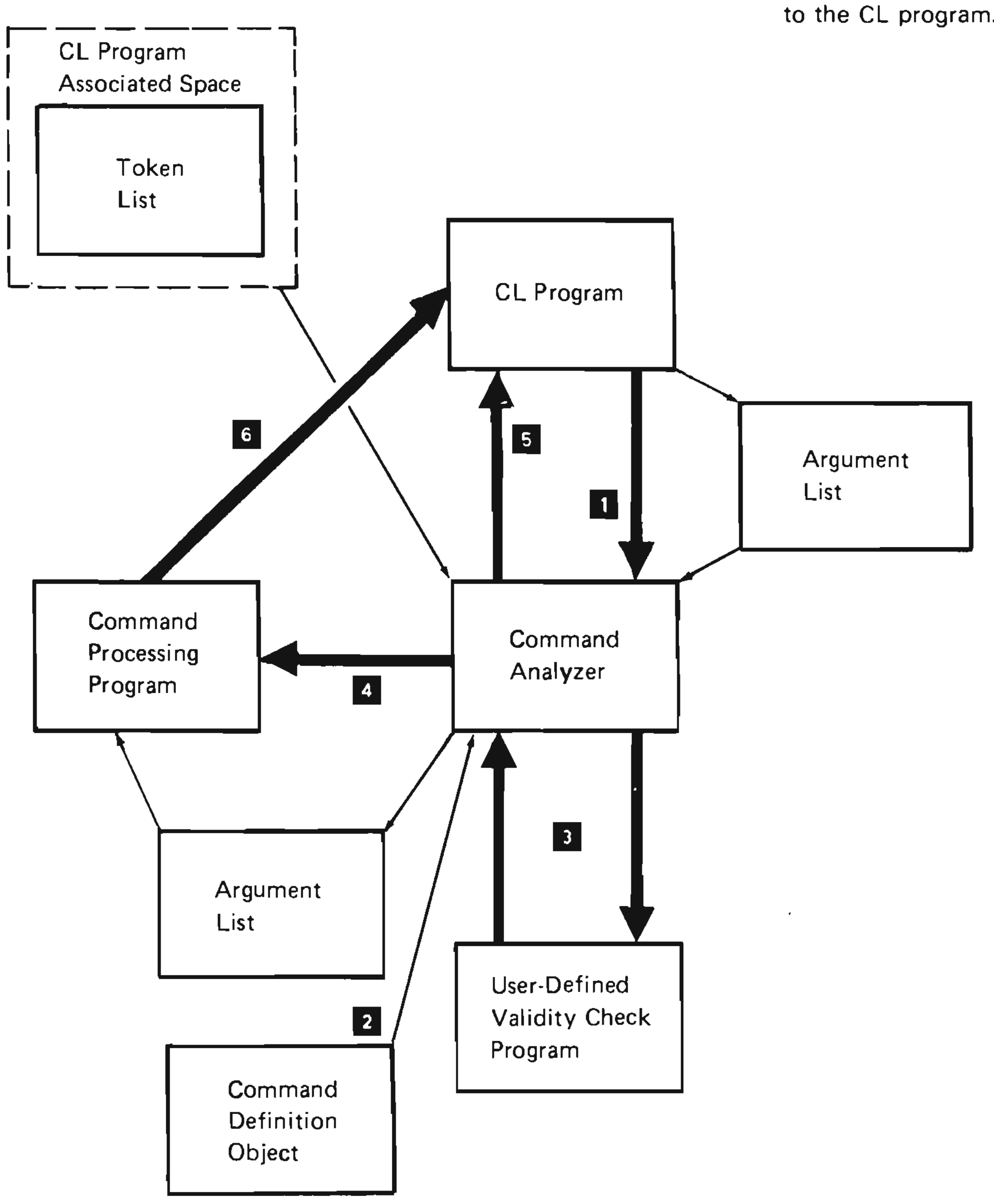


Figure CA-4. Command Analyzer as Used by a CL Program

Command Analyzer as Used by a CL Program with Prompting

Figure CA-5 and the following text describe how the command analyzer is used by a CL program with prompting.

1 The CL program invokes the command analyzer. A token list, which resides in the associated space of the CL program, is passed.

2 A positional list and work area are built, and control is transferred to the prompter, passing this positional list and work area.

3 The prompter calls the command analyzer to validity check the command being entered and to execute the command if it meets the validity check requirements.

4 Control is returned to the prompter if an error is detected and an exception is signaled during command validation, or to enter additional command data.

3 and 4 are repeated until the command is completely entered and properly validity checked.

5 If specified in the CDO, the user-defined validity check program is called to perform extended validity checking.

6 Control is transferred to the proper command processing program, passing an argument list containing pointers to the converted data values in the work area.

7 The command processing program returns control to the CL program.

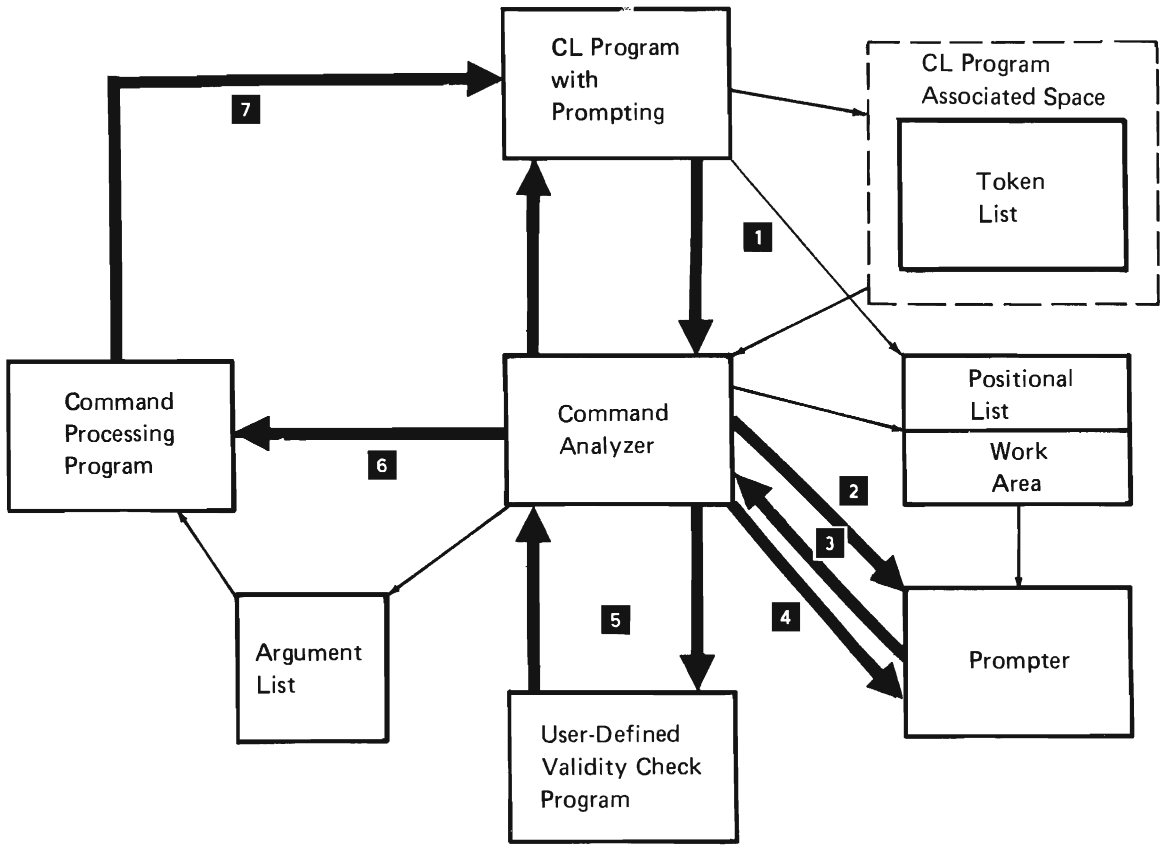

CL Program with Prompting
CL Program Associated Space
Token List
7
1
Command Processing Program
Command Analyzer
Positional List
Work Area
6
2
3
4
5
Argument List
Prompter
User-Defined Validity Check Program

Figure CA-5. Command Analyzer as Used by a CL Program with Prompting

Command Analyzer as Used by the Subsystem Controller without Prompting

Figure CA-6 and the following text describe how the command analyzer is used by the subsystem controller without prompting.

1 The subsystem controller receives a command and calls the command analyzer. An argument list containing a pointer to the positional list space is passed. The positional list contains a pointer to the work area, the option bytes, and the return level. The work area contains the command string and the length of the command string.

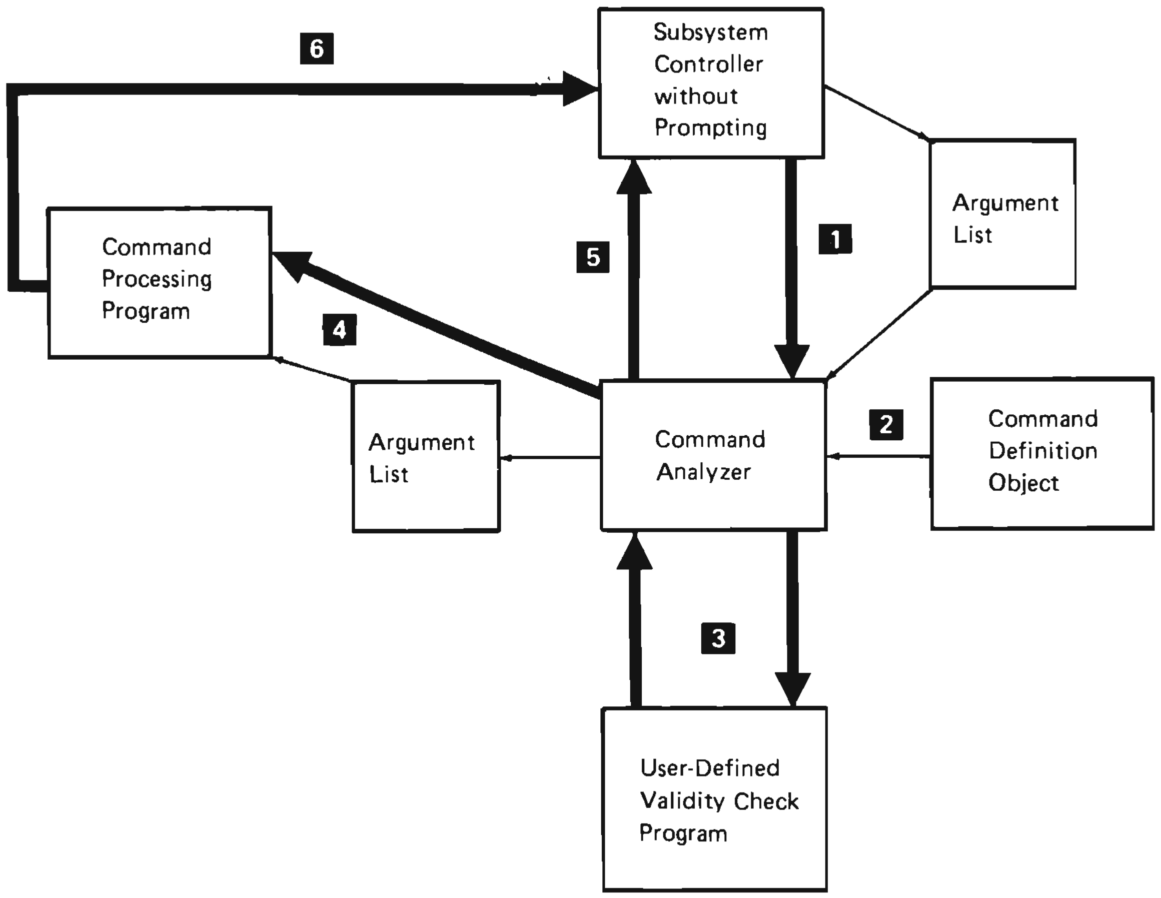
2 The command analyzer, using information in the CDO, validity checks the command.

3 If specified in the CDO, the user-defined validity check program is called to perform extended validity checking.

4 Control is transferred to the proper command processing program. An argument list is passed that contains pointers to the converted data values in the work area.

5 Control is returned to the subsystem controller if a null command is found, or an error is detected and an exception is signaled.

6 Control is returned to the subsystem controller after completion of the application program or command processing program.



6
Subsystem Controller without Prompting
Argument List
1
Command Processing Program
5
4
Argument List
Command Analyzer
2
Command Definition Object
3
User-Defined Validity Check Program

Figure CA-6. Command Analyzer as Used by the Subsystem Controller without Prompting

Command Analyzer as Used by a High-Level Language Program

Figure CA-7 and the following text describe how the command analyzer is used by a high-level language program.

1 A compiled high-level language program invokes the command analyzer by calling the high-level language interface, QCAEXEC. An argument list is passed that contains the command string and its length. QCAEXEC creates space for a work area and a positional list and moves the command string and its length into the work area.

2 QCAEXEC calls QCADRV, passing the work area and positional list. QCADRV and other command analyzer modules, using information in the CDO, validity check and analyze the command, complete the positional list, and convert the data values in the work area.

3 If specified in the CDO, the user-defined validity check program is called to perform extended validity checking.

4 Control is transferred and an argument list containing pointers to the converted data in the work area is passed to the proper command processing program.

5 Control is returned to QCAEXEC if a null command or an error is detected and an exception is signaled.

6 Control is returned to QCAEXEC after completion of the application program or command processing program.

7 QCAEXEC returns control to the high-level language program.

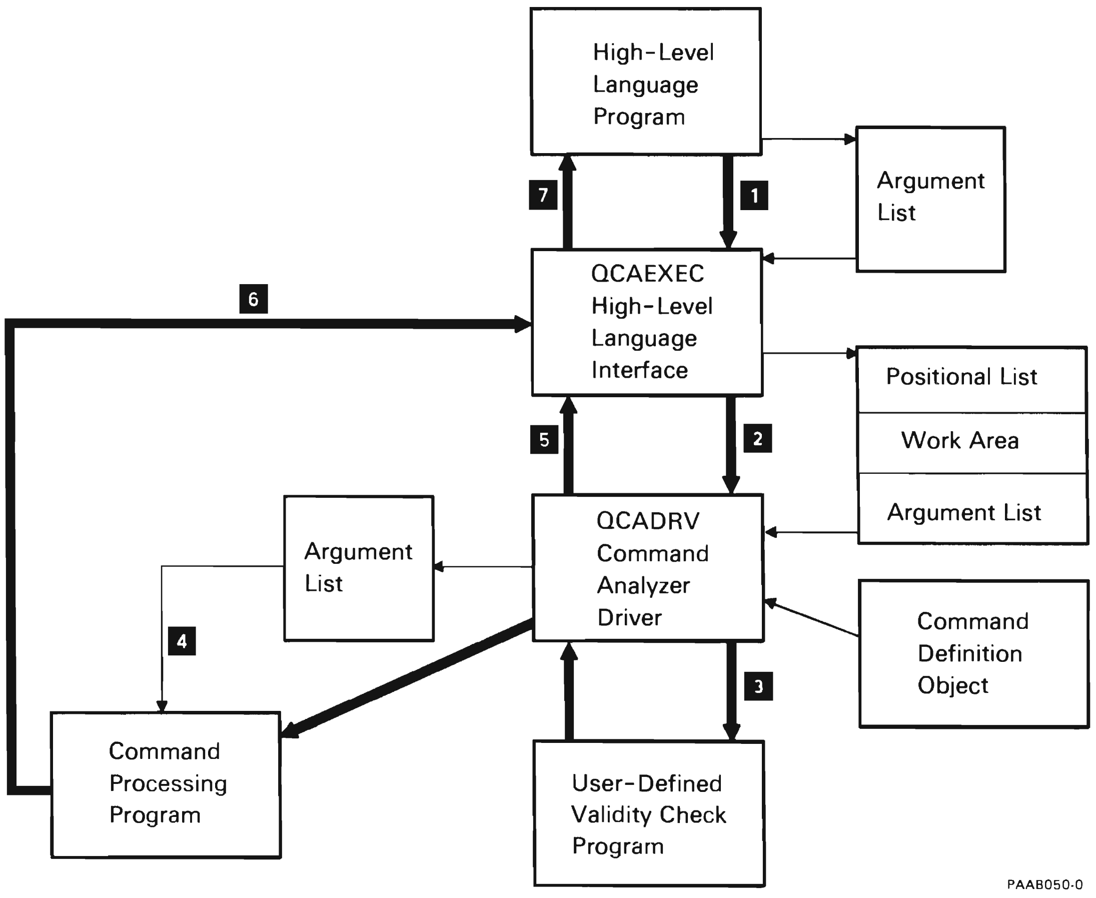

High-Level Language Program
Argument List
7
1
QCAEXEC High-Level Language Interface
6
Positional List
Work Area
Argument List
5
2
QCADRV Command Analyzer Driver
Argument List
4
Command Definition Object
3
Command Processing Program
User-Defined Validity Check Program
PAAB050-0

Figure CA-7. Command Analyzer as Used by a High-Level Language

Command Analyzer as Used by the Subsystem Controller with Prompting

Figure CA-8 and the following text describe how the command analyzer is used by the subsystem controller with prompting.

1 The subsystem controller receives a command and calls the command analyzer, passing a positional list space that contains a pointer to a work area. The work area contains the command character string and length of character string.

2 The command analyzer, using information contained in the CDO, validity checks the command.

3 If the command is not valid, the positional list is not built, an exception is signaled, and control is returned to the subsystem controller.

4 If the command is valid, a positional list is built containing an entry for the command name. Control is transferred to the prompter passing the positional list and work area. Only scalar validity checking is performed; no interparameter checks or user-defined validity checking has occurred. The prompter will prompt for missing keywords, values, or entry errors.

5 The prompter calls the command analyzer with prompted data in the positional list. The command analyzer validity checks the individual scalar values and performs interparameter checks as requested by the prompter. Errors are returned to the prompter.

4 and **5** are repeated until the command is completely entered and properly validity checked.

6 Interparameter checks are performed and, if specified in the CDO, the user-defined validity check program is called to perform extended validity checking.

7 If the command analyzer was called to only validity check the command, as in the case of entering command parameters one at a time, or an error was found in the command, control is returned to the prompter.

8 If the command analyzer was called to execute the command and no errors were found, control is transferred to the proper command processing program. An argument list containing pointers to the converted data values in the work area is passed.

9 The command processing program returns control to the subsystem controller.

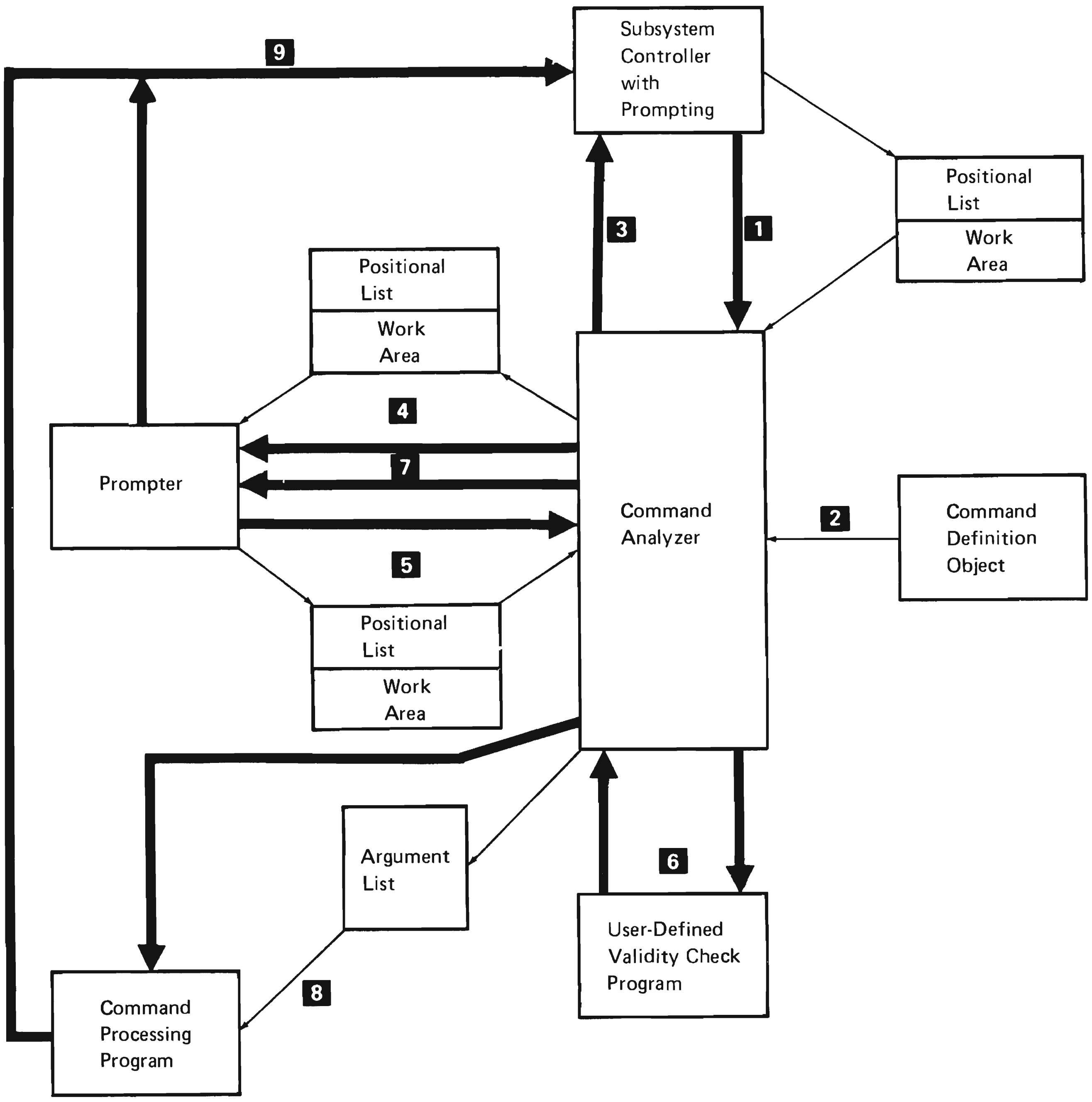

Subsystem Controller with Prompting
Positional List
Work Area
Positional List
Work Area
Prompter
Command Analyzer
Command Definition Object
Positional List
Work Area
Argument List
User-Defined Validity Check Program
Command Processing Program
1
2
3
4
5
6
7
8
9

Figure CA-8. Command Analyzer as Used by the Subsystem Controller with Prompting

INTRODUCTION

The command definition component of the CPF (control program facility) provides command processing programs for the following CL (control language) commands:

- Create Command (CRTCMD)
- Change Command (CHGCMD)
- Display Command (DSPCMD)
- Delete Command (DLTCMD)

Note: The command definition component owns the command syntax, but the librarian component supplies the command processing program for the Delete Command command, the generic delete module QLIDLOBJ.

Using these commands, along with their associated command definition source statements, users can create, change, display, or delete their own commands. This is accomplished by creating, changing, displaying, or deleting the CDO (command definition object) of the specified command.

Command Definition Source Statements

The command definition source statements and their functions are as follows:

- Command statement–CMD: This statement defines the prompt text to be associated with the command being defined. There must be one and only one CMD statement in the source file referred to by the Create Command command.
- Parameter statement–PARM: This statement defines the attributes of command parameters. The order of the PARM statements in the source file specifies the order in which the parameters are passed to the CPP and validity check routine. At least one PARM statement must precede all element, qualifier, or dependent statements. There can be a maximum of 75 PARM statements associated with a command.
- Element statement–ELEM: This statement defines the attributes of elements in a list. If a command parameter consists of a list of elements that are of a different type, each element in the list must be described by an element statement.
- Qualifier statement–QUAL: This statement defines qualified names. If a parameter or list element is a qualified name, that qualified name must be described by QUAL statements.
- Dependent statement–DEP: This statement defines which parameters are dependent on each other. The presence or absence of a parameter and the value of a parameter determine dependency.

Command Definition Objects

Each command has its own unique CDO (command definition object). The CDO contains information about the command. Figure CD-1 shows the contents of a CDO.

1 The description of the parameters is in the form of a binary tree (see Figure CD-2, which follows). There is a node for each PARM (parameter), ELEM (element), and QUAL (qualifier) statement. The parameters are linked together in order; specifically:

- Parameters are linked together in the order they were specified in the CDS (command definition source) statement.
- A PARM node may specify a link to a QUAL or ELEM
- An ELEM node may specify a link to a QUAL or ELEM
- ELEMs of a list are linked together
- QUALs of a qualified name are linked together
- A QUAL may not specify a link to another QUAL or ELEM at a lower level. For example:

```
QUAL TYPE(Q1)
Q1  : QUAL
```

There is also a linked list in which each node in the list represents interparameter dependencies.

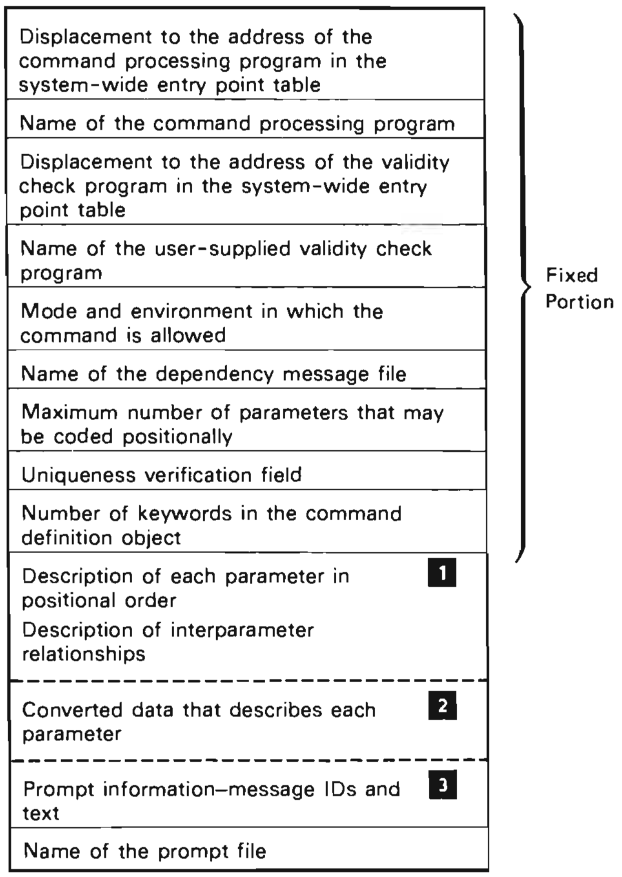
2 Each of the nodes in the binary tree and linked list have displacements to converted data that is used to describe each parameter.

3 Each node in the binary tree has a displacement to the prompt information associated with that node.

Example:

Command definition statements:

	PARM TYPE (E1)	List of two elements
	PARM TYPE (*CHAR)	Scalar
	PARM TYPE (E2)	List of three elements
E1:	ELEM TYPE (*INT2)	List element
	ELEM TYPE (*INT2)	List element
E2:	ELEM TYPE (*CHAR)	List element
	ELEM TYPE (Q1)	List element qualified name of three qualifiers
	ELEM TYPE (*CHAR)	List element
Q1:	QUAL TYPE (*NAME)	Qualifier
	QUAL TYPE (*NAME)	Qualifier
	QUAL TYPE (*NAME)	Qualifier



Displacement to the address of the command processing program in the system-wide entry point table

Name of the command processing program

Displacement to the address of the validity check program in the system-wide entry point table

Name of the user-supplied validity check program

Mode and environment in which the command is allowed

Name of the dependency message file

Maximum number of parameters that may be coded positionally

Uniqueness verification field

Number of keywords in the command definition object

Fixed Portion

Description of each parameter in positional order **1**

Description of interparameter relationships

Converted data that describes each parameter **2**

Prompt information–message IDs and text **3**

Name of the prompt file

Figure CD-1. Command Definition Object

The following is a binary tree representation of the preceding statements:

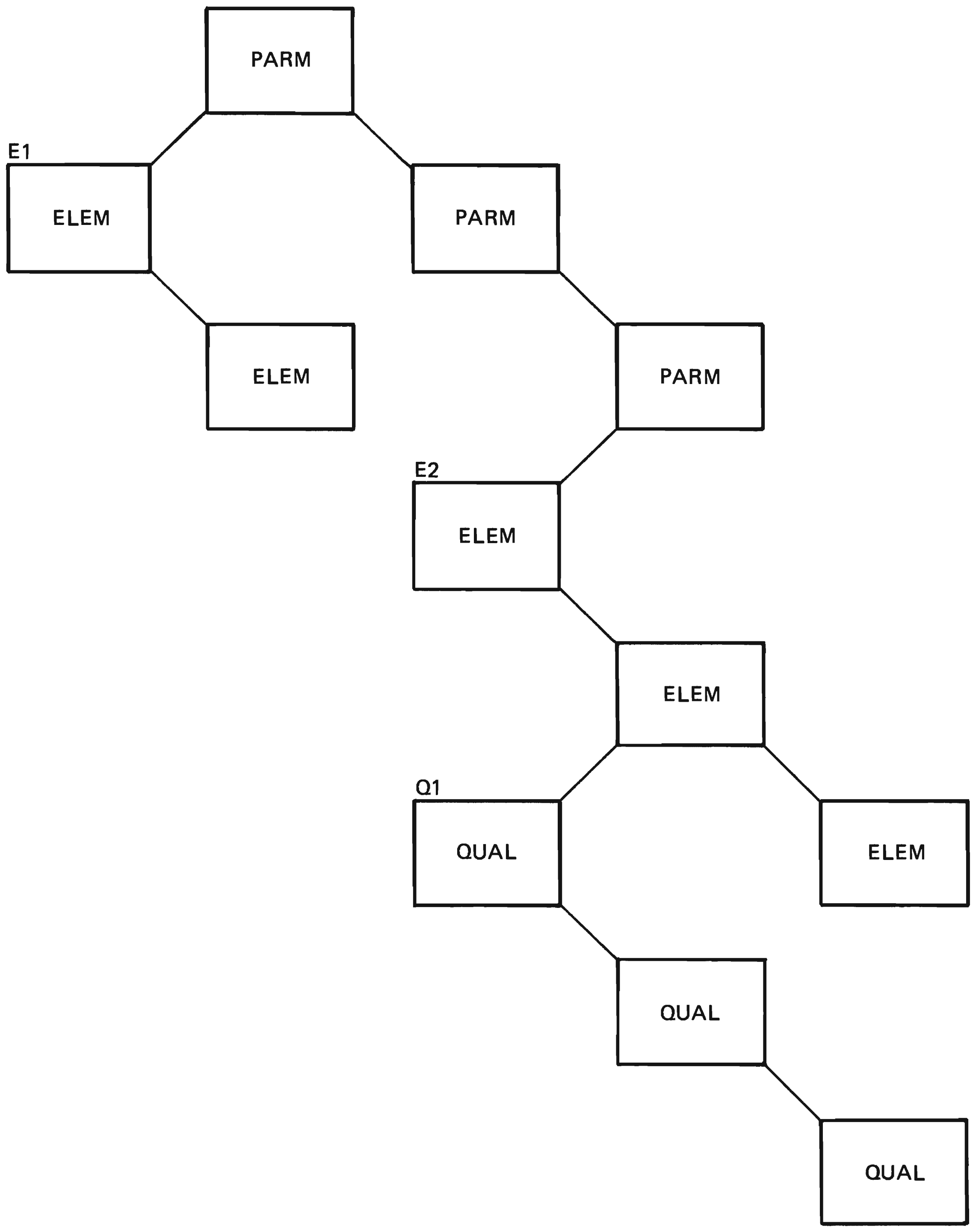


Figure CD-2. Example of Command Parameters in a Binary Tree Format

CDOs, as created or changed, are used by the following CPF components:

- Command analyzer–to validity check commands, to identify the command processing program associated with a command, and to determine the format of parameters passed to that command processing program.
- CL compiler–to identify the name of the command processing program to be called and the format of the parameters passed to that command processing program.
- Data management–to determine the parameter name and prompt text for use by the Display Override (DSPOVR) command.
- Prompter–to determine the prompt text and the format of the parameters used in prompting for command parameters.
- Menu–to determine the command prompt text for use in the all-commands menu display.
- Security–to determine that the user is authorized to the command function that is requested from a display.

Building a Command Definition Object

Figure CD-3 shows an example of how command definition source statements are used to build a CDO.

1 The command definition source statements describe a command that has three parameters, A, B, and C. Parameter A is defined to be one of three restricted character values; each value can be four characters long. Parameter B must be a decimal value in packed decimal 5,2 format and must be a value between 1 and 500. Parameter C is an integer 2 value with a default value of *ONE. *ONE is a special value that maps to an integer value of 1.

2 QCDRPAS1 puts information about the parameters into the CDO. The data describing the parameters is put into a data table. The displacements to parameter information in the data table are also put into the CDO.

3 QCDRPAS2 takes the information in the data table, converts the data into the type specified by the command definition source statements, and puts it at the end of the CDO. The displacement values in the CDO are changed to reflect the displacements to the parameter information that is now in the CDO.

Command Definition Source Statements

1

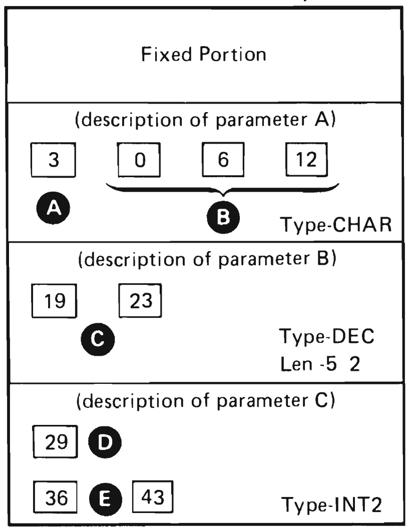
```
LABEL:   CMD
         PARM KWD(A) TYPE(*CHAR) LEN(4) +
               RSTD(*YES) VALUES (BOB JIM MARK)
         PARM KWD(B) TYPE(*DEC) LEN(5 2) RANGE(1 500)
         PARM KWD(C) TYPE(*INT2) DFT(*ONE) SPCVAL((*ONE 1))
```

2

Command Definition Object and Data Table as Built by QCDRPAS1

- **A** Number of restricted values
- **B** Displacements into the Data Table to the restricted values
- **C** Displacements into the Data Table to the lower and upper range values
- **D** Displacement into the Data Table to the default value
- **E** Displacements into the Data Table to the special FROM and TO values

Command Definition Object



Fixed Portion
(description of parameter A)
3
0
6
12
A
B
Type-CHAR
(description of parameter B)
19
23
C
Type-DEC
Len -5 2
(description of parameter C)
29
D
36
E
43
Type-INT2

Data Table (unconverted)

	Attribute	Length	Value
0	CHAR	03	BOB
6	CHAR	03	JIM
12	CHAR	04	MARK
19	NUMERIC	01	1
23	NUMERIC	03	500
29	CHAR	04	*ONE
36	CHAR	04	*ONE
43	NUMERIC	01	1

3

Command Definition Object as Built by QCDRPAS2

- **A** Number of restricted values
- **B** Displacements into the CDO to the restricted values
- **C** Displacements into the CDO to the lower and upper range values
- **D** Displacement into the CDO to the default value
- **E** Displacements into the CDO to the special FROM and TO values
- **F** BIN(15) length followed by character data
- **G** Value in packed decimal (5 2) format
- **H** BIN(15) Integer 2 value

Command Definition Object

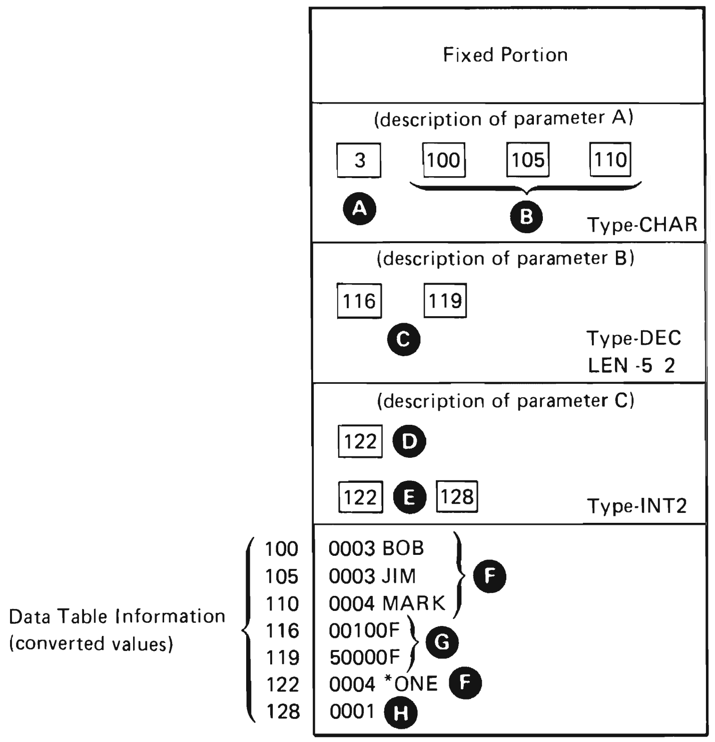


Figure CD-3 Command Definition Object Build Overview

GENERAL OVERVIEW

Command Definition Modules

The command definition component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QCDCCMD–Change Command (CHGCMD)[1]: This module is used to modify the attributes of an existing CDO. The following attributes of a command can be changed:
- CPP name
- Validity checker program name
- Command mode
- Command environment
- Text information

-->QCDDCMD–Display Command (DSPCMD)[1]: This module is used to display or print the following information specified on the Create Command (CRTCMD) command:
- Qualified command name
- Qualified command processing program name
- Qualified source file name (if data base source file)
- Source file member name (if data base source file)
- Qualified validity checking program name
- Valid modes for the command that is displayed
- Valid environments for the command that is displayed
- Maximum number of parameters that may be coded positionally
- Qualified prompt message file name
- Qualified DEP message file name
- Text associated with the command that is displayed

-->QCDRCMD–Create Command (CRTCMD)[1]: This module, along with the following modules, read command definition source statements from the specified data base file, device file, or inline file and create a CDO.

QCDRPAS1–Create Command Pass One: This module reads the command definition source statement file and builds a CDO. QCDRPAS1 also calls the command analyzer to syntax check the command definition source statements.

QCDRPAS2–Create Command Pass Two: This module scans the CDO, validity checks all entries, converts data into an internal format, and moves the converted data to an area near the end of the CDO.

-->QLIDLOBJ–Delete Command (DLTCMD)[1]: This module is used to remove a command from the library in which it resides. It uses the librarian generic delete function command processing program to delete a command. Only the CDO is removed. The validity check program, the command processing program, and the CDS (command definition source) statements are not deleted.

QCDRPAS3–Create Command Pass Three: This module links the parameters in the CDO into the order in which the parameters are to be prompted. The prompt information (text and message identifiers) is then placed at the end of the CDO.

Create Command Command Overview

Figure CD-4 and the following text describe a Create Command (CRTCMD) command overview.

1 The command analyzer decodes the Create Command command, processes it, and transfers control to the command definition module QCDRCMD. A parameter list is passed that contains the Create Command command parameters.

2 QCDRCMD calls QCDRPAS1 to build a CDO and unconverted data table using the information contained in the command definition source statements.

3 QCDRPAS1 calls the command analyzer to validity check the command definition source statements. The command analyzer returns control to the command definition module.

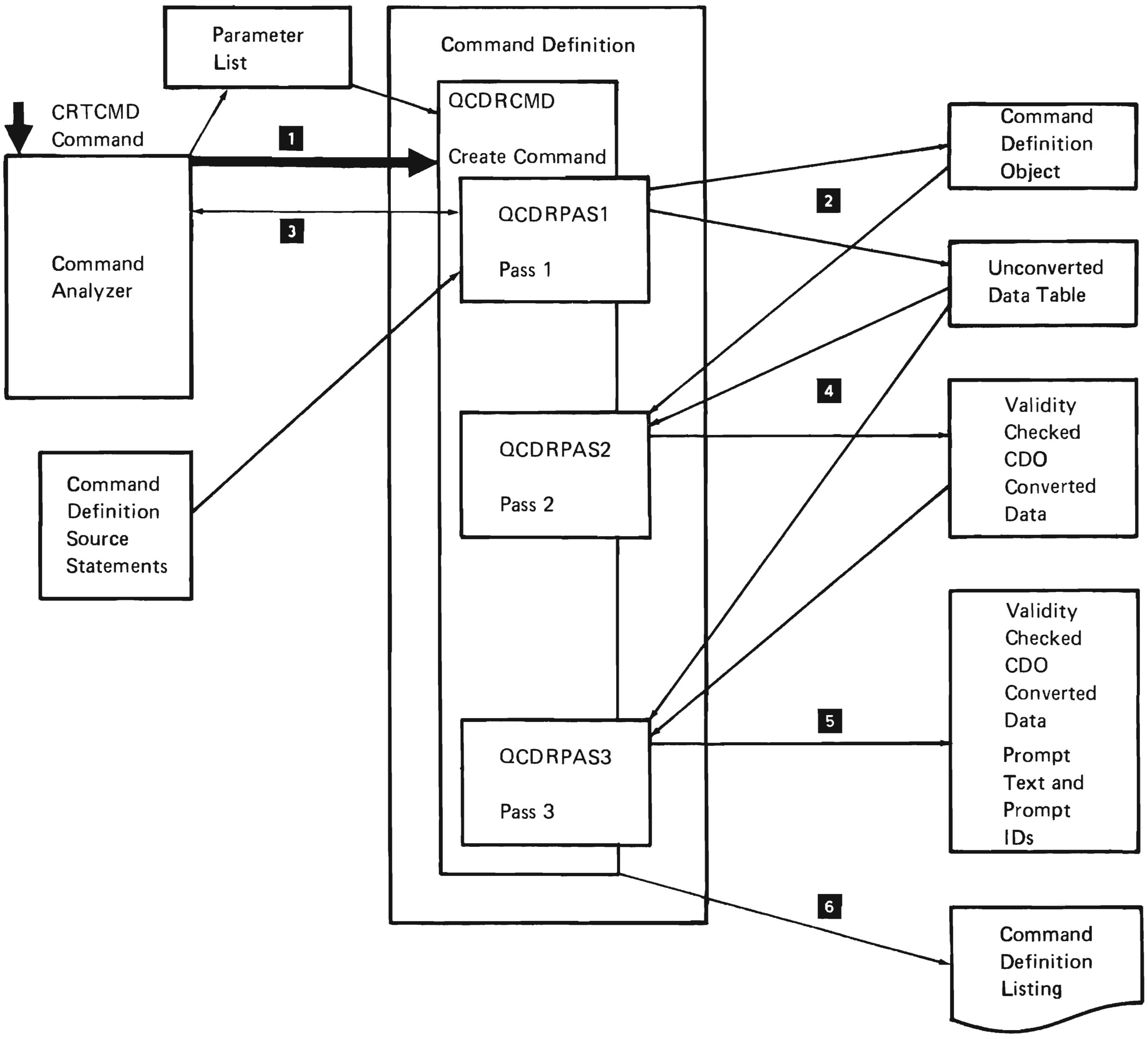
[1]This module is a CPP (command processing program).

4 QCDRCMD calls QCDRPAS2 to validity check the information in the CDO and to convert the data in the table into an internal format. The converted data is moved to an area near the end of the CDO.

5 QCDRCMD calls QCDRPAS3 to link the parameters in the CDO into the order in which they are to be prompted and puts the prompt information at the end of the CDO.

6 QCDRCMD produces a command definition listing that consists of the following:

- Listing of the source statements
- Cross reference listing
- Error messages, if any



Parameter List
CRTCMD Command
Command Analyzer
Command Definition Source Statements
Command Definition
QCDRCMD
Create Command
QCDRPAS1
Pass 1
QCDRPAS2
Pass 2
QCDRPAS3
Pass 3
1
2
3
4
5
6
Command Definition Object
Unconverted Data Table
Validity Checked CDO Converted Data
Validity Checked CDO Converted Data Prompt Text and Prompt IDs
Command Definition Listing

Figure CD-4. Create Command Command Overview

Control Language Compiler

INTRODUCTION

The CL (control language) compiler component of the CPF (control program facility) provides the command processing programs for the Create Control Language Program (CRTCLPGM) command, Retrieve Job Attributes (RTVJOBA) command, List Command Usage (LSTCMDUSG) command, Convert Date (CVTDAT) command, Retrieve Control Language Program Source (RTVCLSRC) command, Retrieve Data Area (RTVDTAARA) command, and Display Program (DSPPGM) command.

The Create Control Language Program (CRTCLPGM) command is the CL compiler. The CL compiler performs the following functions:

1 Reads CL commands from a source file.

2 Performs intracommand and intercommand validity checking. The command analyzer is used for individual command syntax checking and parsing. A positional list and work area is passed.

3 Generates the code for an object program. The generated code is called an IRP (intermediate representation of a program).

4 Calls the PRM (program resolution monitor) to generate a machine interface template from the IRP.

5 Produces a listing of all commands read from the source file, along with any errors and cross-reference listings of labels and variables.

Figure CL-1 shows the relationship among the CL compiler and the command analyzer and program resolution monitor during execution of the Create Control Language Program command.

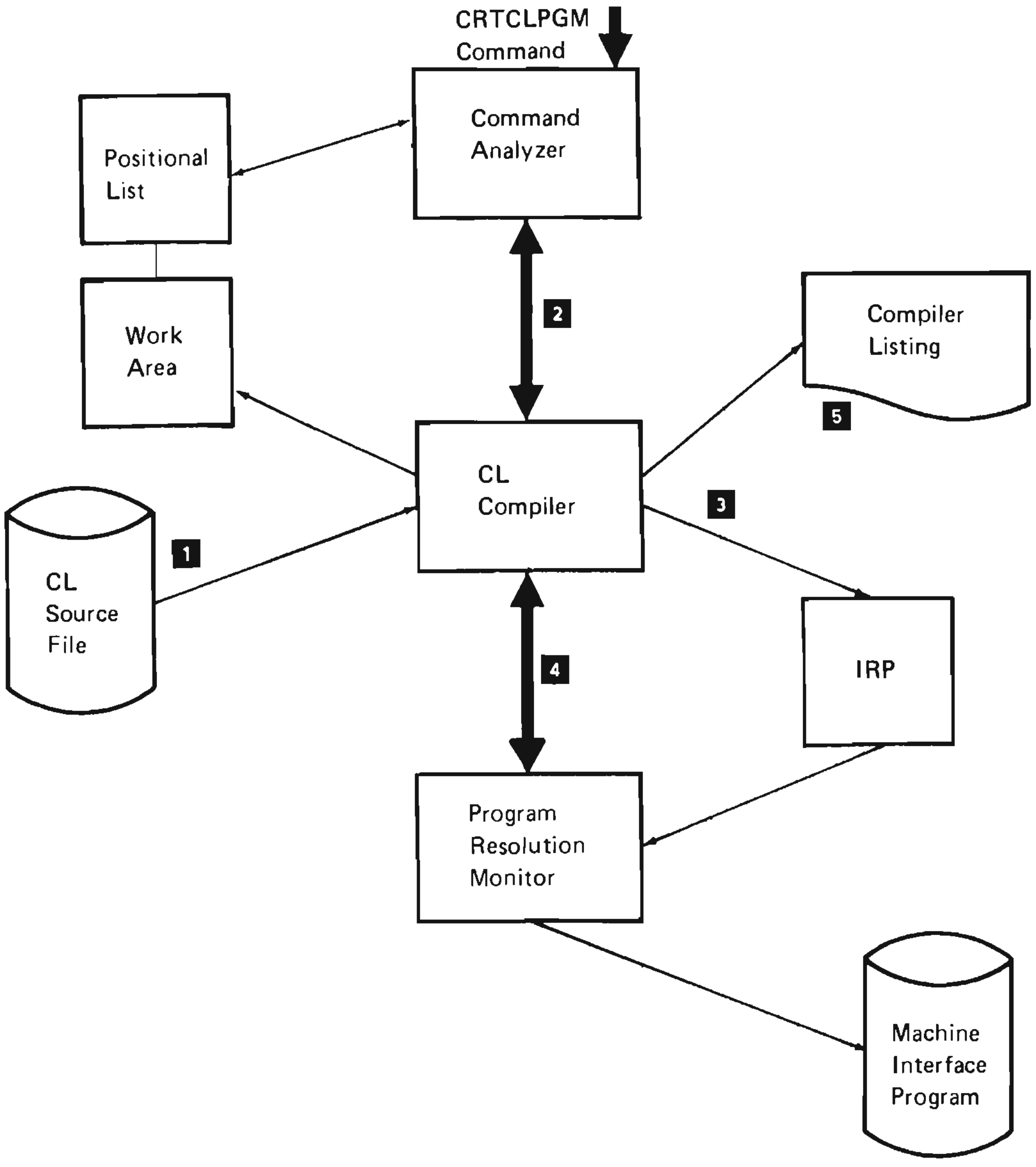


Figure CL-1. CL Compiler System Overview

GENERAL OVERVIEW

CL Compiler Modules

The CL Compiler component consists of the following modules:

Note: An arrow (-->) identifies a module as an entry module into the component. Indentation of a module shows its dependency on a previous module.

Compile-Time Modules

-->QCLENTR–CL Compiler Driver: This module is the entry to the CL compiler. It performs the following functions:

- Processes the Create Control Language Program command keywords
- Calls other CL modules to perform initialization, process source commands, and create the program
- Handles terminal exceptions occurring during compilation
- Prints the compiler listing from the compiler print space
- Destroys compiler tables
- Closes files

QCLINIT–Initialization: This module creates spaces and indexes needed by the CL compiler. It also initializes the communications area used by CL modules, opens the files used during compilation, and reads all source into the compiler source space.

QCLCMDPR–Command Processing: This module retrieves source commands from the compiler source space, puts them in the compiler listing space, and calls the command analyzer to parse and syntax check them. It calls other compiler modules to process commands and to do cleanup work. QCLCMDPR contains the main processing loop for the compiler. QCLCMDPR also puts diagnostic messages produced by the command analyzer and lower-level CL compiler modules into the compiler print space. This module saves source information for inclusion in the program's associated space.

QCLICP1–Initial Command Processing 1: This module handles labels and invokes the compile-time processor for the Declare CL Variable command, Declare Data Area command, Declare File command, and Program command. This module saves command information for inclusion in the program's associated space.

QCLDSVST–Declare Data Area and Build Symbol Table: This module processes the Declare Data Area command.

QCLDECST–Declare and Symbol Table Build: This module processes the Declare CL Variable command.

QCLDMDCL–Data Manipulation Declare: This module processes the Declare File command. It also produces a list of variable names and attributes for variables implicitly declared by the Declare File command.

QCLPRCMD–Program Command: This module processes the Program command.

QCLICP2–Initial Command Processing 2: This module handles the Do, Enddo, Return, and End Program commands. It also generates code for the end of the THEN clause of an If command, the end of the ELSE clause, and the end of the MONMSG EXEC clause. This module saves command information for inclusion in the program's associated space.

QCLGTCMD–Goto Command: This module processes Goto commands in CL programs.

QCLSSVRC–Send Data Area: This module processes the Send Data Area command.

QCLIFCMD–If Command: This module processes If commands in CL programs.

QCLGNEVL–Generate Evaluation Code: This module processes expressions for the If and Change Variable commands in CL programs.

QCLBIFCN–Built-In Functions: This module processes the built-in functions.

QCLRSVRC–Receive Data Area: This module processes the Receive Data Area command in CL programs.

QCLMNMSG–Monitor Message Command: This module processes the Monitor Message command in CL programs.

QCLXCCMD–Transfer Control: This module processes the Transfer Control command in CL programs.

QCLDMWC–Data Manipulation Wait/Cancel: This module processes the Cancel Receive and Wait commands in CL programs.

QCLDMCMD–Data Manipulation Commands: This module processes the Send File, Receive File, and Send Receive File commands in CL programs.

QCLCLCMD–Call Command: This module processes Call program commands in CL programs.

QCLCHVAR–Change Variable: This module processes the Change Variable command in CL programs.

QCLGNEVL–Generate Evaluation Code: This module processes expressions on the Change Variable command.

QCLBIFCN–Built-In Functions: This module processes the built-in functions.

QCLREGCL–Regular Commands: This module processes all commands that are *not* compiled inline.

QCLWUIS–Where Used Information Save: This module analyzes the commands processed by QCLREGCL, and saves information about any value (that the CDO for the command indicates is a file). This information is saved in a structure for inclusion in the OIR for the CL program.

QCLLSCMD–Else Command: This module processes the Else command in CL programs.

QCLDMFIN–Data Manipulation Finish: This module generates the following tables:
- UFCB (user file control block)
- Data manipulation table containing pointers to all data tables, corresponding record format names, and other I/O information
- Data manipulation tables, one table for each record format name that was referred to in the program

This module saves information from the data manipulation table for inclusion in the program's associated space.

QCLSTOUT–Symbol Table Output: This module declares variables in IRP format, puts variables, labels, and any corresponding attributes and cross-references in the CL program listing.

This module saves information from the symbol and parameter tables for inclusion in the program's associated space.

QCLMSFIN–Monitor Message Code Completion: This module processes tables built by QCLMNMSG. It generates code to declare, enable, and disable message monitors in a CL program.

QCLIRPC–IRP Completion: This module generates the epilogue code for all CL programs. This module formats saved command information for inclusion in the program's associated space.

QCLCRPGM–Create Program: This module sets CL compiler attributes that, when passed to the PRM will be used in defining and creating a CL program. QCLCRPGM then invokes the PRM to create the CL program. This module copies information saved during compilation into the created program's associated space.

QCLCMPXH–Compiler Exception Handler: This module is called by other CL modules to handle the space-addressing violation exception during compilation.

The following module is used by QCLCLCMD, QCLXCCMD, QCLDMCMD, and QCLDMWC:

QCLMAP–Resolution of Special Value Mappings: This module extracts a special FROM value from the CDO on commands that are compiled to inline code when a variable is coded on a keyword with special value mapping.

The following module is used by QCLICP1, QCLICP2, QCLWUIS, QCLREGCL, QCLDMCMD, and QCLDMWC:

QCLERROR–Error routine: This module is called by compile-time modules to send diagnostic and escape messages.

Execution-Time Modules

-->QCLRSLV–Resolve: This module is called at the start of every CL program. It performs the following functions:
- Allocates a positional list and work area, if needed.
- Checks to see if the CL program is compatible with the current release/modification level.
- Checks to see if the command logging is in effect for the current execution of the CL program.

-->QCLCLCPR–Call (CALL,TFRCTL)[1]: This module is called to handle the interface to an application program. QCLCLCPR also performs logging for the compiled Call and Transfer Control commands. For the Transfer Control command, QCLCLNUP is invoked to perform cleanup before the CL program relinquishes control.

-->QCLDMIO–Data Manipulation I/O (SNDF, RCVF, SNDRCVF, CNLRCV, WAIT)[1]: This module performs the functions of the Send File, Receive File, Send/Receive File, Cancel Receive, and Wait commands. This module also performs the logging functions for these commands.

-->QCLRTNE–Return Execution Module: This module is invoked by the CL program on a normal return. QCLRTNE performs logging for a compiled Return command and normal end-of-program logging functions. This module invokes QCLCLNUP to perform cleanup for a CL program.

-->QCLCLNUP–Clean-up: This module is invoked when the invocation of a CL program is ended by exception processing. QCLCLNUP deallocates the space objects of the program, and closes the data manipulation files used by the program. This module is also invoked by QCLCLCPR and QCLRTNE to perform clean-up.

-->QCLRSVRE–Receive Data Area (RCVDTAARA)[1]: This module processes the Receive Data Area command, and performs the logging function for the Receive Data Area command.

-->QCLSSVRE–Send Data Area (SNDDTAARA)[1]: This module processes the Send Data Area command, and performs the logging function for the Send Data Area command.

-->QCLRTVJA–Retrieve Job Attributes (RTVJOBA)[1]: This module processes the Retrieve Job Attributes command.

-->QCLXCEXC–Transfer Control Exception Handler: This module receives control if the code generated for a CL program Transfer Control command gets a machine exception. QCLXCEXC then invokes QCLRSLV, which creates work spaces needed by the program, sends an appropriate CPF message, and signals a CPF exception. This causes the call and transfer control diagnostic interfaces to appear consistent to the user.

-->QCLCMXRF–List Command Usage (LSTCMDUSG)[1]: This module processes the List Command Usage command.

-->QCLCNVNC–Convert Numeric to Character: This module is invoked by CL programs to perform numeric to character conversions, when required by the Change Variable command.

-->QCLCNVCN–Convert Character to Numeric: This module is invoked by CL programs to perform character to numeric conversions, when required by the Change Variable command.

-->QCLCNVDT–Convert Date (CVTDAT)[1]: This module processes the Convert Date command.

-->QCLXERR–Execution Time Error Handler: This module is invoked when a function check occurs in a CL program which does not have a Monitor Message command to handle the function check. QCLXERR sends an inquiry message and processes the reply.

[1]This module is a CPP (command processing program).

-->QCLXDUMP–Dump CL Program (DMPCLPGM)[1]: This module processes the Dump CL Program command. QCLXDUMP is invoked by QCLXERR when a dump is requested.

-->QCLRTVDA–Retrieve Data Area (RTVDTAARA)[1]: This module processes the Retrieve Data Area command.

-->QCLSCAN–Scan for a Pattern in a String: This module, called from a high-level language program, finds the position where a character string occurs within another string of characters.

Control Language Command with Independent Command Processing Program

Figure CL-2 shows the relationship between a CL command with an independent command processing program, the command analyzer, and the command processing program.

1 The CL program invokes the command analyzer. A token list, which resides in the associated space of the CL program, is passed.

2 The command analyzer performs the validity checking and interfaces with the prompter, if necessary. If the command analyzer detects any errors, it signals an exception to the CL program. If no errors are detected, control is transferred to the command processing program.

If logging is requested, the command string is rebuilt in a keyword format and sent to the calling CL program as a command message. The command then appears in the job log.

3 The command processing program performs its function and returns control to the CL program.

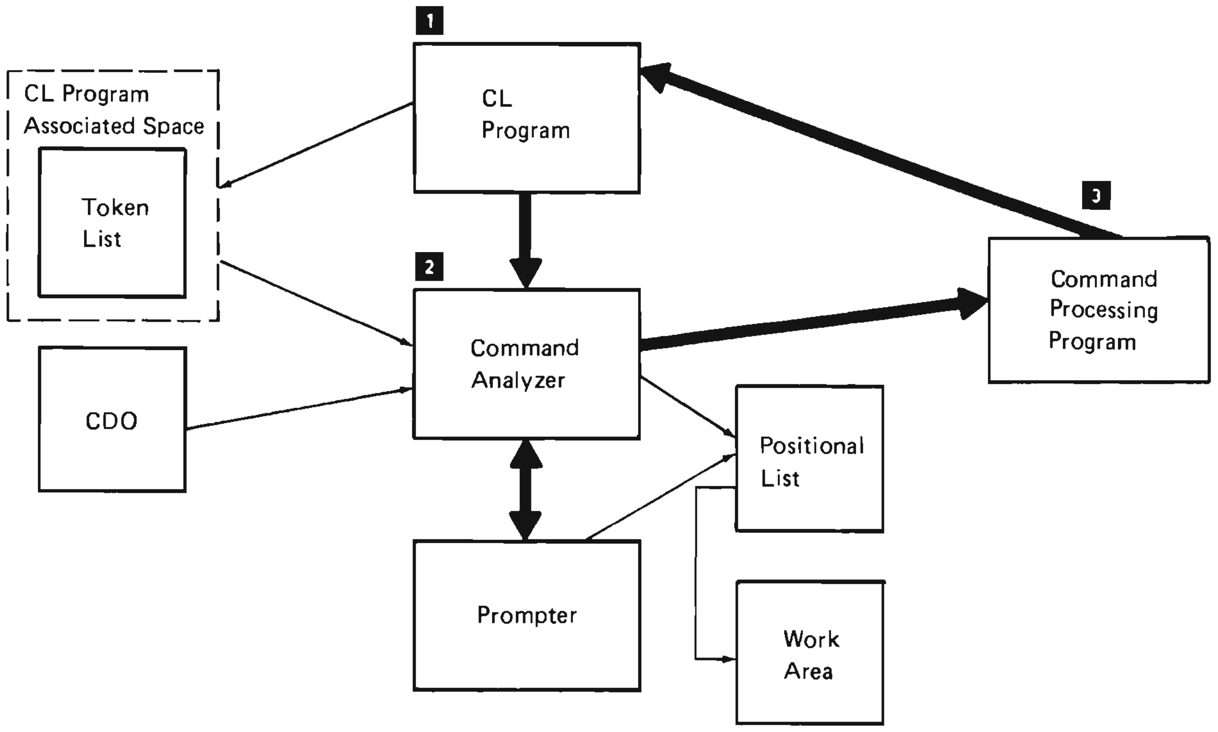


Figure CL-2. Control Language Command with Independent CPP Overview

[1]This module is a CPP (command processing program).

CL Program Call and Transfer Control Commands

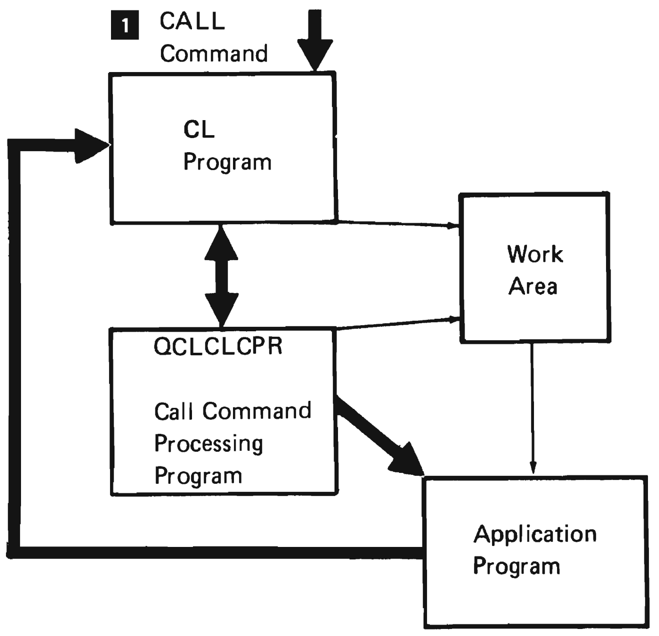
Figure CL-3 shows an overview of the operation of the CL Call (CALL) and Transfer Control (TRFCTL) commands.

Call Command

1 The CL program initializes a control block, initializes a work area, and calls QCLCLCPR. If the control language program does not have a pointer to the application program, QCLCLCPR resolves addressability to the application program. QCLCLCPR sets up an argument list and transfers control to the application program. If the program name or library name was coded as a variable in the CL program, QCLCLCPR validity checks the contents of the variables for type name. QCLCLCPR also translates any machine interface exceptions on the call or the resolve of addressability to command processing program exceptions for the CL program. If logging is requested, QCLCLCPR logs the Call command.

Transfer Control Command (Generic Description)

2 If the CL program does not have a pointer to the application program, QCLCLCPR resolves addressability to the application program for the CL program. If logging is requested, QCLCLCPR logs the Transfer Control command. The CL program then transfers control to the application program.



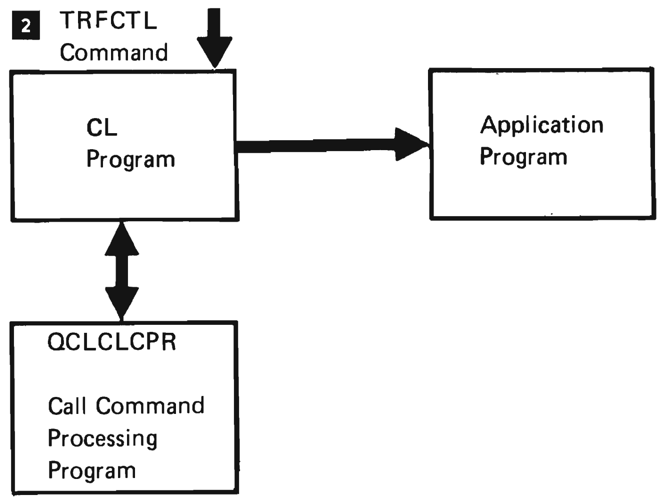


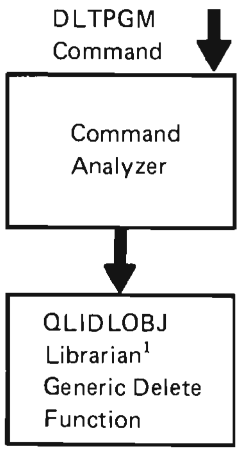
Figure CL-3. CL Program Call and Transfer Control Command Overview

Delete Program Command

The Delete Program (DLTPGM) command deletes a program from the specified library along with the corresponding OIR (object information repository) data. See Figure CL-4 for the Delete Program command overview.

Retrieve Job Attribute Command

The Retrieve Job Attribute (RTVJOBA) command retrieves certain job attributes and puts them in a CL variable. QCLRTVJA extracts the values requested by the Retrieve Job Attribute command from the work control block, process attribute template, and job message queue, and returns these values to the CL program. See Figure CL-5 for the Retrieve Job Attribute command overview.



[1] See the librarian component (LI) for further detail.

Figure CL-4. Delete Program Command Overview

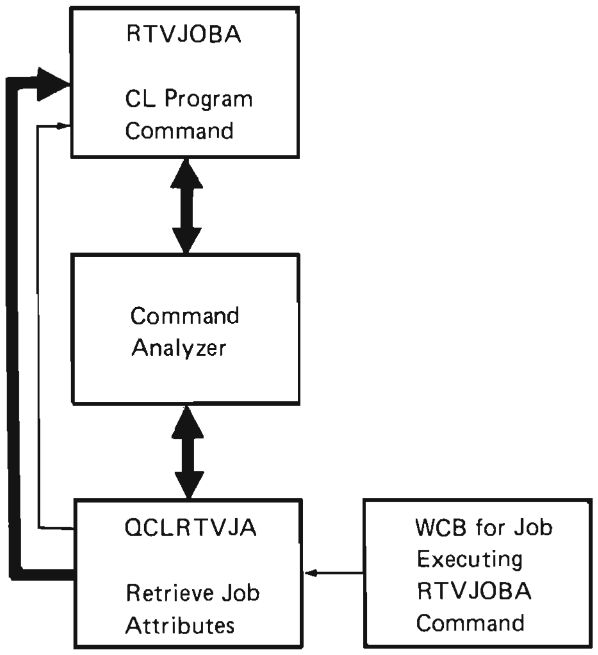


Figure CL-5. Retrieve Job Attribute Command Overview

Convert Date Command

The Convert Date (CVTDAT) command converts a date from one format to another. Figure CL-6 and the following text describe the operation of the Convert Date command.

1. QCLCNVDT calls QWCSVRDR to verify that the date is correct.

2. QCLCNVDT calls QWCSCDFR to convert the date to the proper format.

3. QCLCNVDT returns the converted date to the CL program.

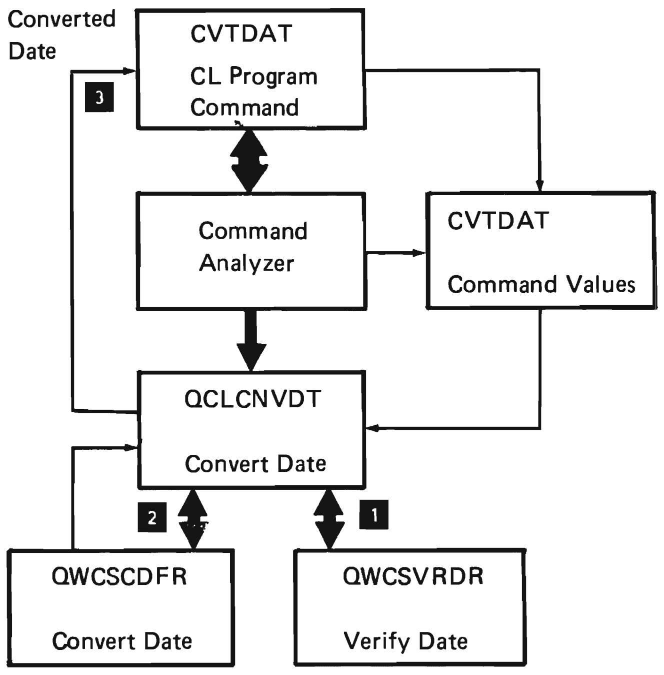


Figure CL-6. Convert Date Command Overview

List Command Usage Command

The List Command Usage (LSTCMDUSG) command generates a report showing which CL programs use one or more of the commands entered on the List Command Usage command. Figure CL-7 and the following text describe the operation of the List Command Usage command.

1. QCLCMXRF calls the librarian module QLILIST to obtain a list of programs matching the value specified on the List Command Usage command.

2. QCLCMXRF examines each program to determine if it is a CL program.

3. QCLCMXRF examines each CL program in the list to determine if the program uses any of the commands specified on the List Command Usage command.

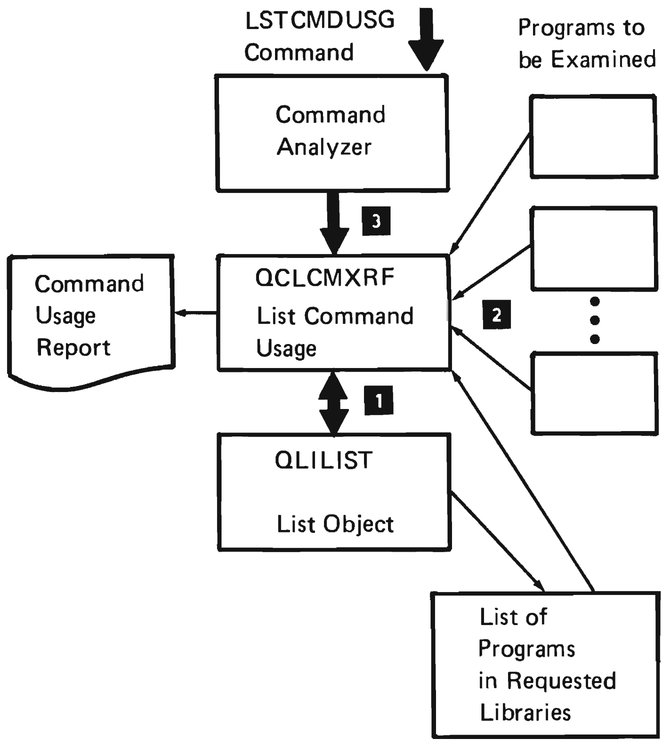


Figure CL-7. List Command Usage Command Overview

Retrieve CL Source Command

The Retrieve CL Source (RTVCLSRC) command retrieves the source and other information about the program, saved in the CL program object, and places it in a data base source file. Figure CL-8 and the following text describe the operation of the Retrieve CL Source command.

1 QCLRTVDR invokes subroutines to perform the retrieve function and sends appropriate escape and completion messages to the caller of QCLRTVDR.

2 QCLRTVVI performs the following:

- Verifies that the program exists and is a CL program.
- Verifies authority to the CL program.
- Locks the CL program.
- Verifies that the file exists, that the file is a data base source file, and that it has a proper record length.
- Adds the member to the file, if necessary.
- Opens and clears the data base source file member.

3 QCLRTVEI performs the following:

- Extracts the CL program's source from the CL program associated space.
- Extracts the patch/no patch option from the CL program associated space.
- Extracts original source file information from the CL program OIR.
- Extracts the user modification flag from the CL program OIR.

4 QCLRTVFS formats the CL program source and other information and puts it in the data base source file member.

5 QCLRTVCU is used in the Retrieve CL Source command processing. QCLRTVCU performs the following:

- Unlocks any programs that are locked.
- Destroys any work spaces.
- Closes any files.

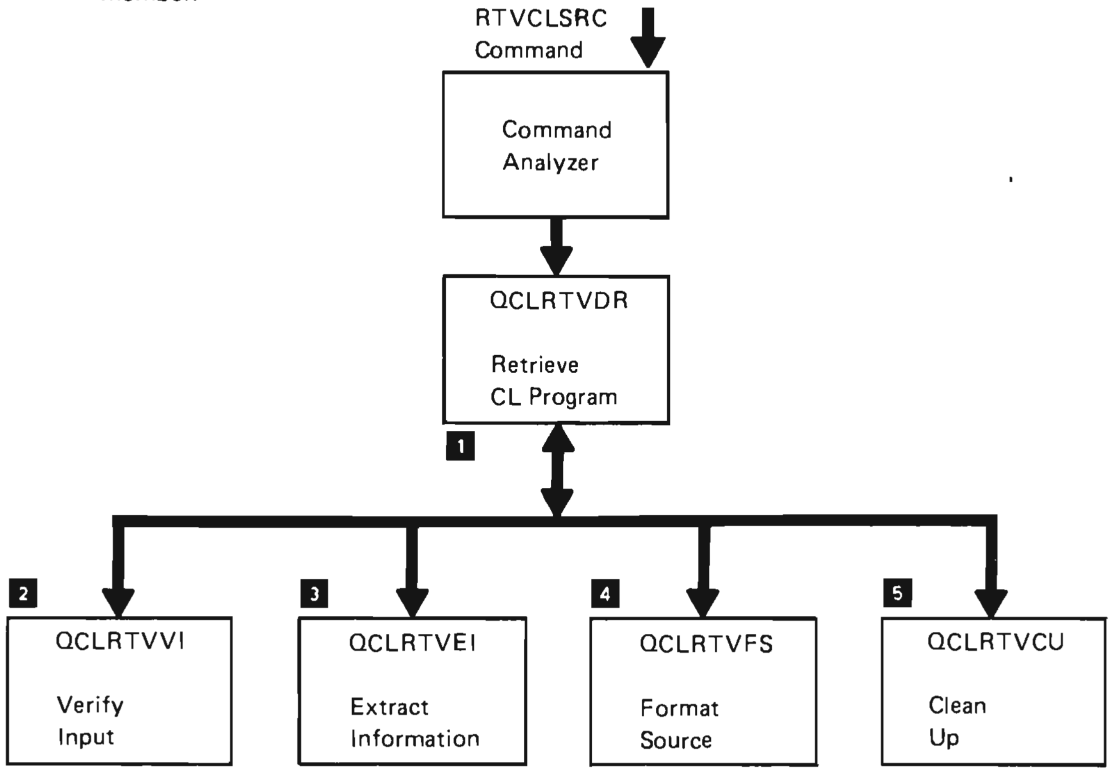


Figure CL-8. Retrieve CL Source Command Overview

Display Program Command

The Display Program (DSPPGM) command produces a display showing the attributes of a program. Figure CL-9 and the following text describe the operation of the Display Program command.

1 The command analyzer decodes a Display Program command, and control is transferred to QCLDSPPG.

2 The program specified on the command is accessed to obtain its attributes.

3 The program attributes are formatted and displayed or printed.

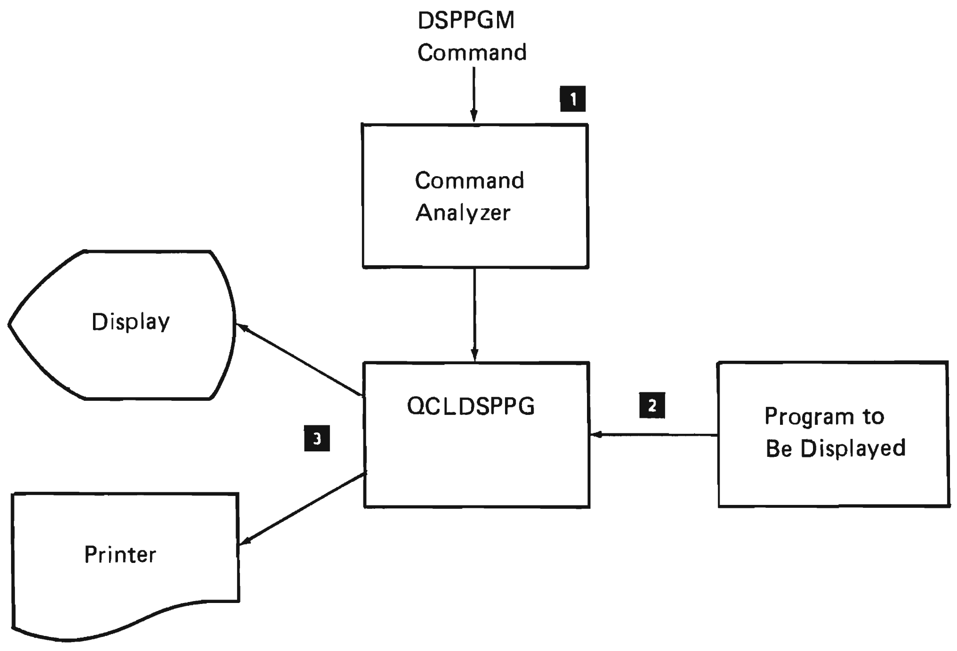


Figure CL-9. Display Program Command

Change Program Command

The Change Program (CHGPGM) command modifies the attributes of one or more programs. Figure CL-10 and the following text describe the operation of the Change Program command.

1 QCLCHGPG calls the librarian module QLILIST to obtain a list of programs matching the value specified on the CHGPGM command.

2 QCLCHGPG makes the changes to each program. If the text is being modified, module QLIMROIR is called to change the text. If the program must be re-encapsulated, a new program object is created, and the old program is replaced with the new program.

3 If any errors are encountered, QCLCHEXT is called as an invocation exit program to back out any changes that are not complete.

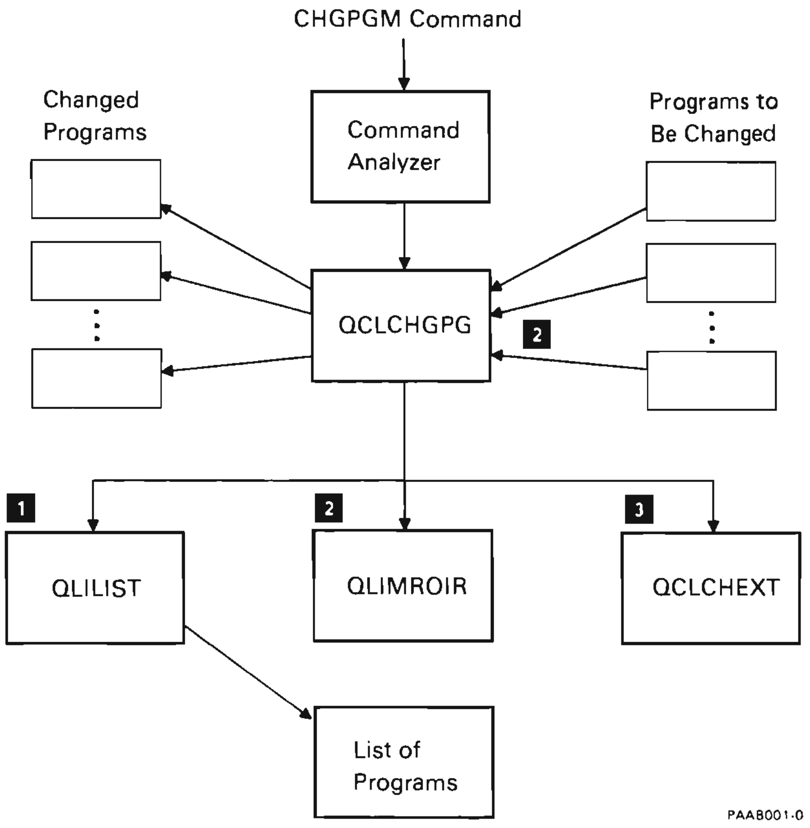


Figure CL-10. Change Program Command Overview

Console Function Manager

INTRODUCTION

The console function manager component of the CPF (control program facility) provides the interface between the console and display operations. This interface causes the console to appear similar to the interface of the 5251 display function manager. Unlike the 5251 function manager, the console function manager uses module QCOREQIO to perform its own I/O operations, instead of the SNA-T3 component.

The console function manager runs independently of processes or devices active on the system, and is capable of running in several processes simultaneously.

GENERAL OVERVIEW

Console Function Manager Modules

The console function manager component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QCOOPEN–Console Open: This module opens a display file.

-->QCOPUT–Put to Display: This module performs the write-to-display function.

> QSFPUT–Put to Subfile: This module puts and updates subfile records and subfile message records.
>
> QSFCRT–Create Subfile: This module creates the subfile and sets up necessary subfile controls.
>
> QCOSFLCT–Console Subfile Control Records Function: This module displays the subfile and performs functions on the subfile as specified in the subfile control record: clear, delete, and so forth.
>
> QSFMQDSP–Message Queue Display: This module handles the program message queue display functions.

-->QCOGET–Console Get Function: This module performs the read from display as well as several operator-requested operations, such as field validation, print display, and second level text.

> QSFGET–Get From Subfile: This module retrieves records from a subfile.
>
> QSFHSFL–Help Key Support: The module prompts QCOGET to display the second level text for a message in a message subfile.
>
> QCORTSFL–Roll/Truncate/Fold Subfile Support: This module performs subfile roll and fold/truncate functions.
>
> QCOVLFLD–Field Validation: This module handles field validation functions.

-->QCOCLOSE–Console Close: This module closes a display file.

-->QCORST–Restore File: This module restores a suspended file on a display device.

-->QCOMEEH–Console Nowait Event Handler: This module handles the machine-signaled event (REQIO complete) that follows a nowait operation.

-->QCOSPEND–Suspend File: This module suspends a file on a display device.

-->QCOMSG–Console Message Function: This module turns the console Message Waiting light on or off.

-->QCOPTMSG–Special Put Message: This module is used by CPF work management to display an operator message independent of the device file open.

-->QCODSMSG–Special Display Status Message: This module is used by CPF to display a message with the status of an active display file.

The following module is used by QCOREQIO.

QCOERROR–Console Exception Handling Routine: This module handles exceptions signaled by QCOREQIO.

The following module is used by QDCCRLUD, QWCISCFR, and QDCVALUD.

QCOLUDIN–LUD Initialization: This module initializes the LUD (logical unit description) associated space for the console and is normally invoked only at CPF start time.

The following module is used by QCOOPEN, QCOPUT, QCOGET, QCOSPEND, QCORST, QCOMSG, QCOCLOSE, QCOPTMSG, QCODSMSG, and QCOMEEH.

QCOREQIO–Request I/O: This module issues the console I/O requests.

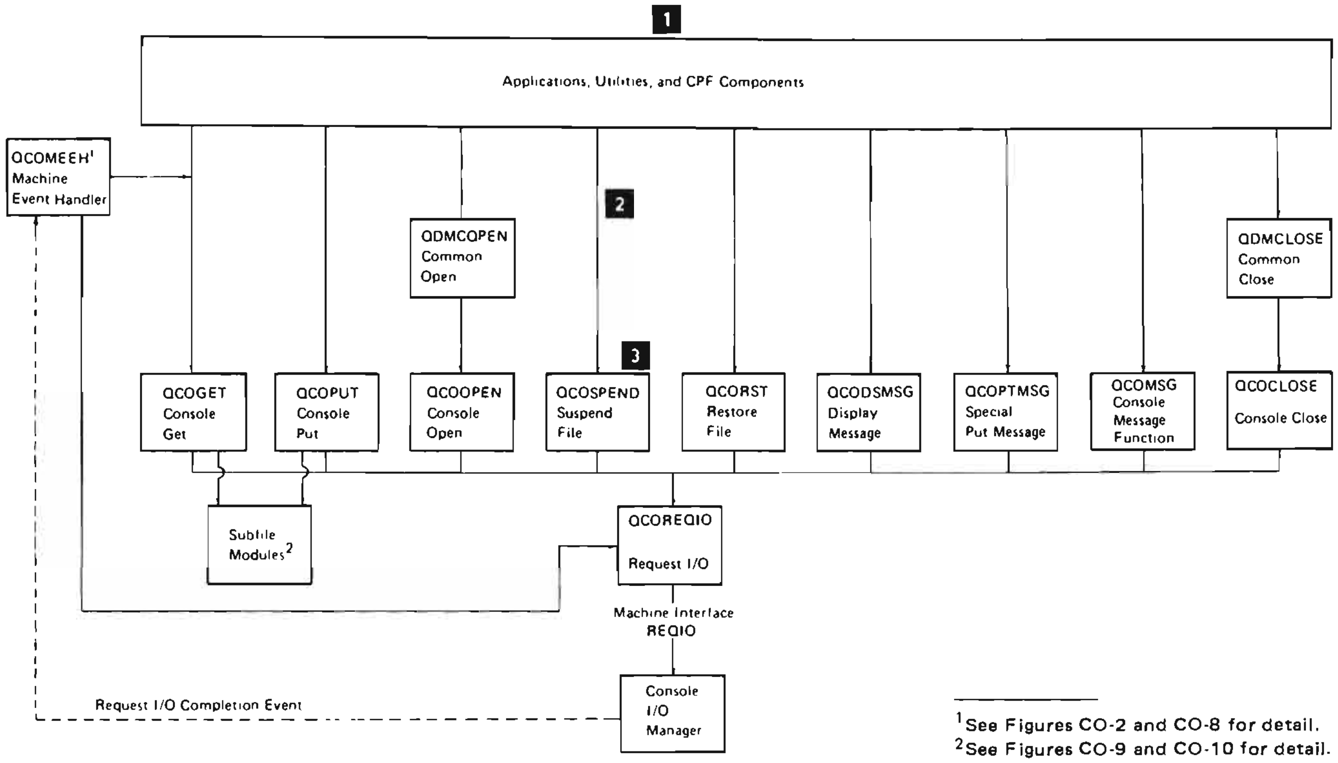
Console Function Manager External Interfaces

Figure CO-1 and the following text describe the external interfaces to the console function manager. An external interface is defined as a call from the using program to a module within the function manager.

1 Function manager users: The function manager does not distinguish one external user from another. A call from an external user identifies a requested function. The function manager performs that function regardless of who the caller is.

2 Upward interface: All execution time requests and controls from the user program are provided through this interface. The program provides information to the modules through the UFCB (user file control block), the option list, the control list, and the user output buffer. Information is returned in several ways: through the ODPCB (open data path control block) feedback areas, the UFCB buffer pointers, events, exceptions, and the user input buffer.

3 Function manager modules: Each call to a console function manager module is specified through an ODP (open data path) to the console display.



¹See Figures CO-2 and CO-8 for detail.
²See Figures CO-9 and CO-10 for detail.

Figure CO-1. Console Function Manager External Interfaces

Console Function Manager Internal Interfaces

Figure CO-2 and the following text describe the internal interfaces to the console function manager.

1 The REQIO complete event is signaled whenever a put nowait or get nowait operation is completed.

2 QCOMEEH dequeues the completed request block from the machine interface response queue and calls QCOREQIO to process the request block. When control returns to QCOMEEH, it then determines if the request type is a put or a get. If the request is a put nowait, the put nowait complete event is signaled to the user.

3 If the request is a get nowait, QCOGET is called to process the input data. The get routines in QCOGET can: detect an operator error, send a message, and reissue the get nowait; handle an operator request–help or print; print user data. If the get routines do process good user input data, QCOMEEH signals a data available event to the user, but that event will not be signaled if the get nowait had to be reissued.

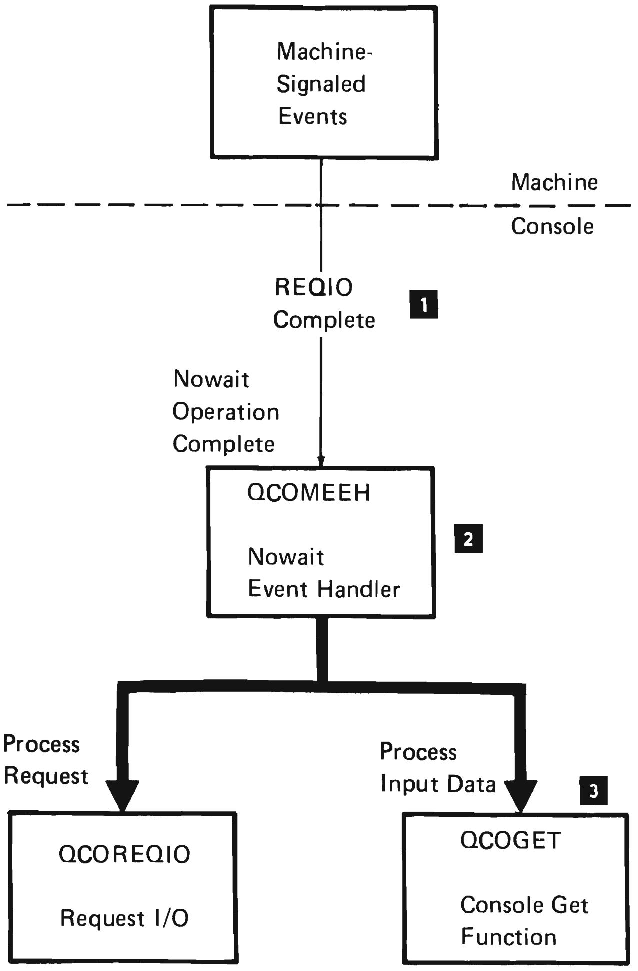


Figure CO-2. Console Function Manager Internal Interfaces

Put Operation

Figure CO-3 and the following text describe a normal put operation.

1 For a field-level put request, the user buffer contains the option indicators followed by all hidden and output fields as described in the device file. Field data is processed one field at a time and is put into the source/sink data area along with the necessary controls to display the data. Hidden fields are saved in the function manager work area.

A nonfield-level put user buffer contains a character string that is treated by the function manager as a single field of data to be sent to the display. The character string is moved from the user buffer to the source/sink data area without being examined or altered. All necessary controls to display the data are supplied by the function manager.

The user buffer for a user-defined data stream request contains the complete console-dependent data stream. As in a nonfield-level put, the data is moved into the source/sink data area just as it was passed to the function manager. With a user-defined data stream request, the function manager does *not* add display controls to the data.

2 Message file: The user can optionally specify a message from the message file as output data. QCOPUT retrieves the message and sends it to the console user.

3 Device file: Default data and constants from the device file can be sent to the console. The data is taken from the device file, processed and placed in the source/sink data area.

4 Source/sink data: The source/sink data (part of the request block) is where the function manager builds the console-dependent data stream that is to be transmitted to the console. The data stream contains all of the controls needed to display the user record as described in **1**.

5 Function manager work area: The function manager work area is a part of the ODPCB (open data path control block). As the function manager builds the output data stream in the source/sink data area, it also builds a user buffer image in the function manager work area. This work area contains an image for each active record on the console screen.

6 Job log: The device file record, as seen in the user buffer, can optionally be sent to the job log as well as to the console.

7 I/O feedback area: This area is updated at various times while the function manager is performing the put operation.

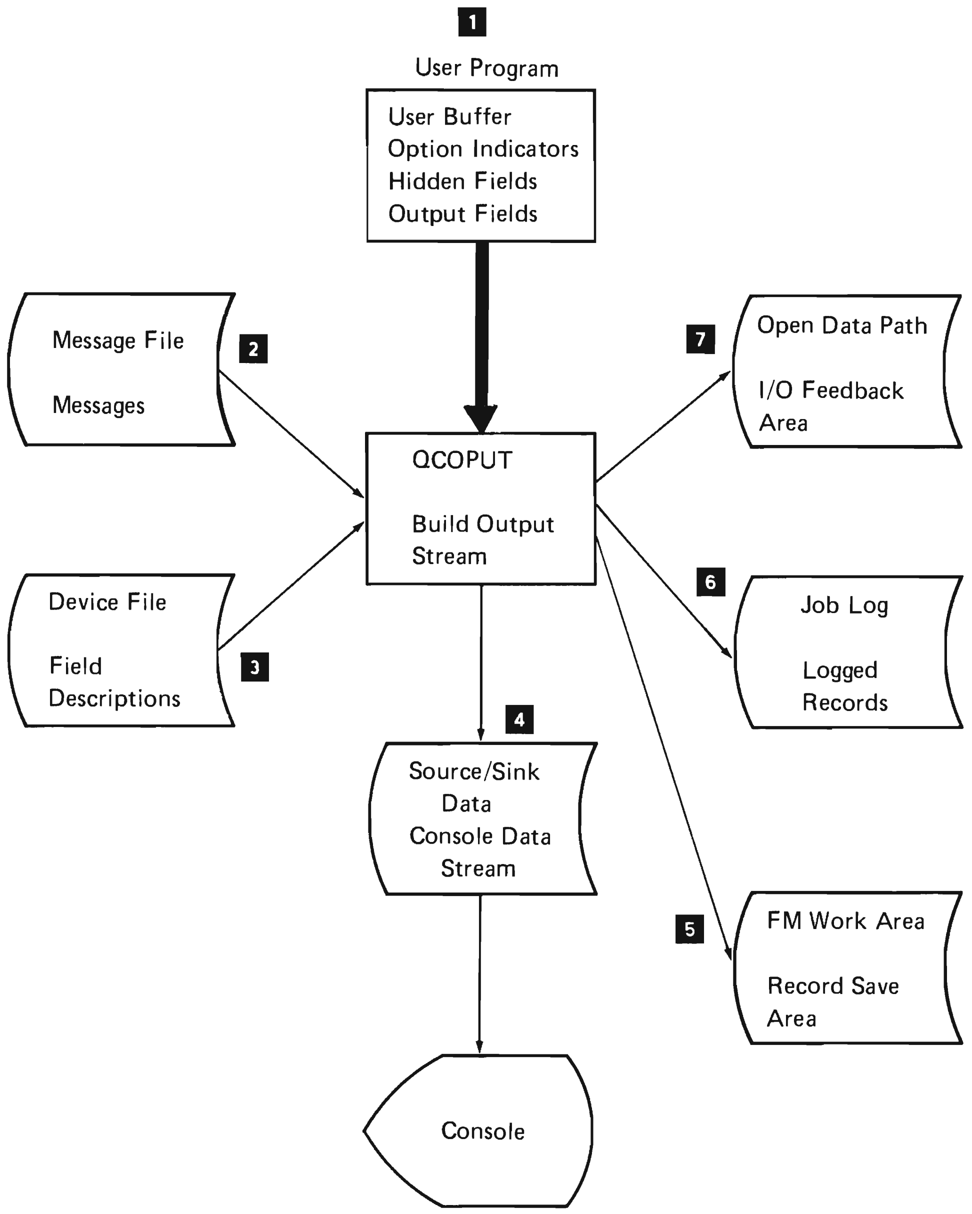


Figure CO-3. Put Operation

Put to Subfile Record (Data Flow)

Only field-level files provide subfile functions. The subfile functions do not support nonfield-level files. User data is treated the same as if the request were to a nonsubfile record by the put routines. QCOPUT determines that the record is a subfile record and transfers control to QSFPUT.

Figure CO-4 and the following text describe a put to a subfile record operation.

1 For a subfile record put request, the user buffer (or a separate indicator area) contains the option indicators followed by all hidden and output fields as described in the device file.

2 The field descriptions for the subfile record are received from the device file and placed in the subfile.

3 QSFPUT takes the data and its attributes, one field at a time, and saves them in the subfile. All of the information needed to display the record as part of the subfile is maintained in the subfile, except for constants that are kept in the device file. The record remains in the subfile until the using program requests that it be displayed.

4 QSFPUT calls QSFCRT during the first subfile record put operation to create the subfile space that will receive all the data from the user buffer or the program message queue if the subfile is a message subfile.

5 The user can optionally specify a message key and a program message queue as a source of data for message subfiles. The subfile modules receive a copy of the messages specified from the message queue and place these into the subfile.

6 QSFPUT calls QSFMQDSP when the put operation to the subfile record specifies that the data to be placed in the subfile should come from a program message queue instead of the user buffer.

7 The subfile is where all subfile information from the user buffer is stored. The subfile record remains in the subfile until the using program requests that it be displayed.

8 The record, as seen in the user buffer, can optionally be sent to the job log as well as to the subfile. This is done independent of the subfile functions.

9 The I/O feedback area is updated at various times while the function manager is performing the put operation. Additional information, such as the relative record operated on, is put in the I/O feedback area for subfile functions.

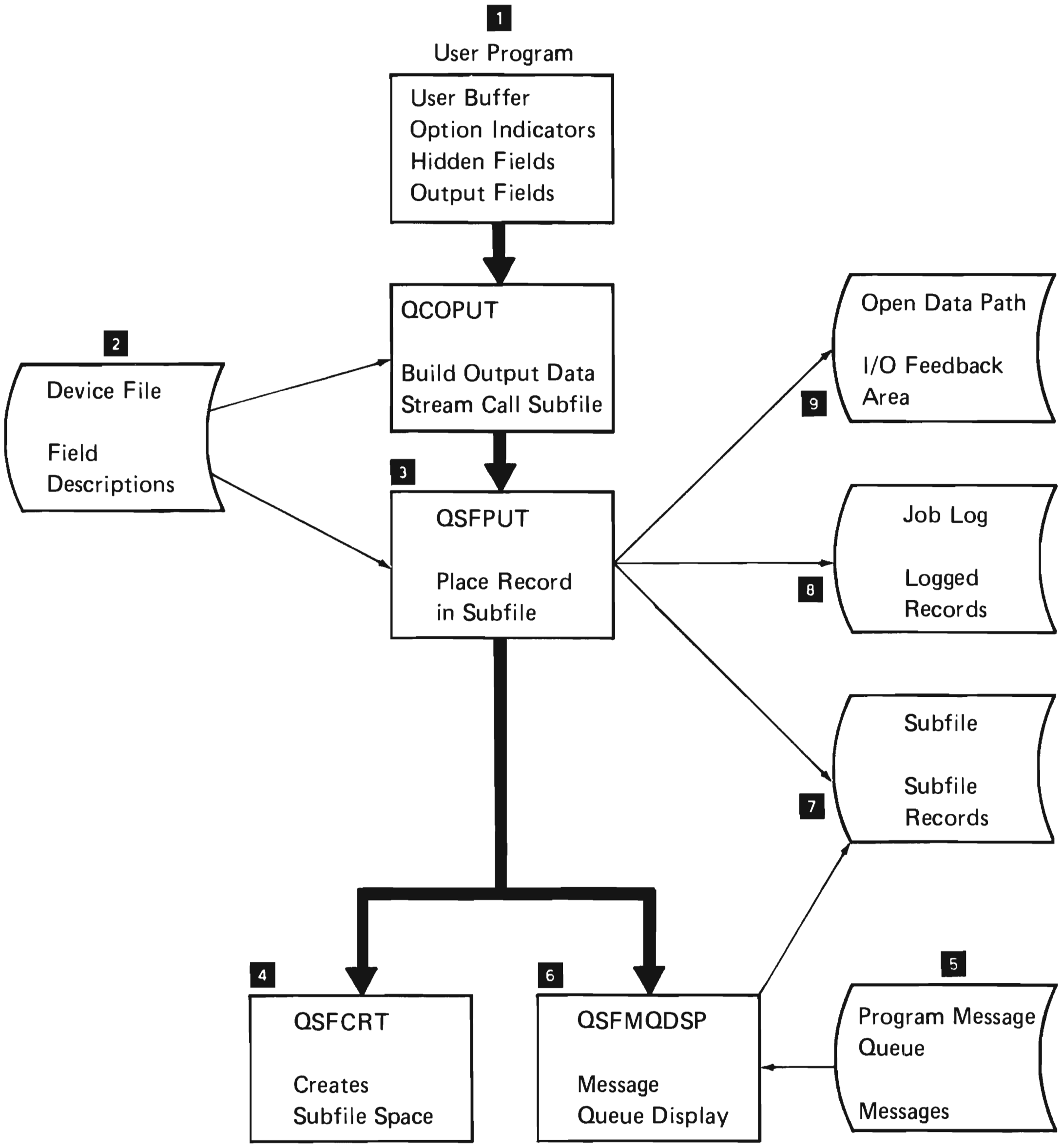


Figure CO-4. Put to Subfile Record (Data Flow)

Put to Subfile Control Record (Data Flow)

Only field-level files provide subfile functions. The subfile functions do not support nonfield-level files. User data is treated the same as if the request were to a nonsubfile record by the put routines. QCOPUT calls QCOSFLCT if a put is done to the subfile control record, and a subfile function is requested (such as initialize, delete, and so forth). Operations to the subfile control record only cause I/O to the console if the display subfile or display subfile control record function is indicated.

Figure CO-5 and the following text describe a put to a subfile control record operation.

1 For a subfile control record put request, the user buffer contains the option indicators followed by all hidden and output fields as described in the device file.

2 The function manager processes a put to the subfile control record in two passes. First, the function manager processes a put to the subfile control record as if it were a nonsubfile record. This includes preparing any fields contained in the subfile control record for display by building a data stream, if the display subfile control record keyword is specified. Second, the function manager checks if the display subfile keyword is specified, and if so calls QCOSFLCT to add the subfile records to be displayed to the data stream.

3 If the subfile initialize keyword is specified for the subfile control record, and the subfile space has not been created, QCOPUT calls QSFCRT to create the subfile space for the subfile records. QCOPUT then calls QCOSFLCT to initialize the subfile records from the device file description.

4 The constants are used from the device file to build subfile records in the data stream.

5 The user can optionally specify a message key and a program message queue as a source of data for a subfile of messages. The subfile modules receive a copy of the messages specified from the message queue and place these into the subfile.

6 The subfile is where all subfile information from the user buffer is stored. The subfile record remains in the subfile until the using program requests that it be dispiayed.

7 The source/sink data, which is a part of the request block, is where the subfile and console function managers build the console-dependent data stream that is to be transmitted to the console. The data stream contains all of the controls needed to display the user record as defined in the device file, and is built only when displaying the subfile via a put to the subfile control record.

8 The function manager work area is a part of the ODPCB (open data path control block). As the function manager builds the output data stream in the source/sink data area, it also builds a user buffer image in the function manager work area. This work area contains as many buffer images as there are records with input capable fields on the console screen, except for subfile records. A single record buffer image is maintained for each subfile description, regardless of the number of records displayed.

9 The record, as seen in the user buffer, can optionally be sent to the job log as well as to the subfile. This is done independent of the subfile functions.

10 The I/O feedback area is updated at various times while the function manager is performing the put operation. Additional information, such as the relative record operated on, is put in the I/O feedback area for subfile functions.

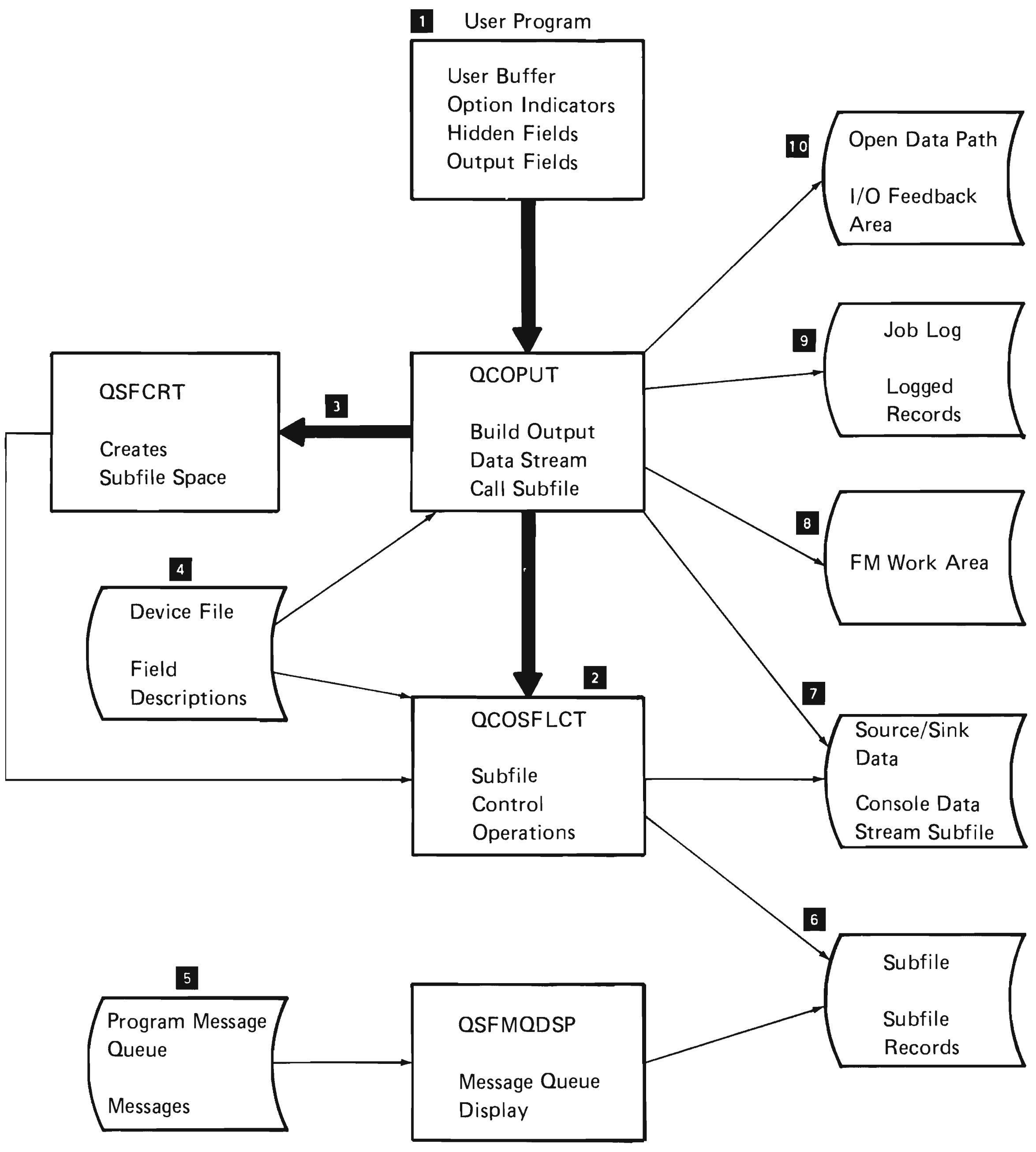


Figure CO-5. Put to Subfile Control Record (Data Flow)

Get Operation

Figure CO-6 and the following text describe a normal get operation.

1 Actual data does not come from the device file during a get operation, but record and field descriptions from the device file are used to process the input information.

2 To initiate the read operation, a Read Modified command is put in the source/sink data area and transmitted to the console. To satisfy the read, the console user must press a command key or the Enter/Record Advance key.

3 When the read operation is completed, all fields with the modified data tag on are returned to the function manager in the same source/sink data area that contained the Read Modified command.

4 If the field description specifies validity checks (such as range or list check) during field processing, QCOGET calls QCOVLFLD to perform those checks.

5 Input records can optionally be logged in the job log.

6 If subfile records are received from the console, QCOGET alters its normal field process to handle the subfile records. When the first subfile field data is received, QCOGET retrieves the proper record from the subfile and places it in the record save area of the function manager work area. All fields received for that subfile record are then processed in the normal manner. When all fields are processed, the subfile record is returned to the subfile with the new data. This process is repeated for each subfile record that is modified by the console user.

7 When QCOPUT builds the output data stream in the source/sink data area, it also builds a user buffer image in the function manager work area. These buffer images are updated as QCOGET processes each modified field returned from the console. At the completion of the get operation, all of the modified data on the display screen will be represented in the record save area of the function manager work area, in the subfile, or in both places.

8 The I/O feedback area is updated at various points in the function manager during the get operation.

9 For the normal field-level get operation, the user buffer contains response indicators followed by all of the input fields (including any hidden fields) as described in the device file. The data is moved from the record save area in the function manager work area to the user buffer.

During a normal nonfield-level or a user-defined data stream get operation, the input data is moved directly from the source/sink data area to the user buffer.

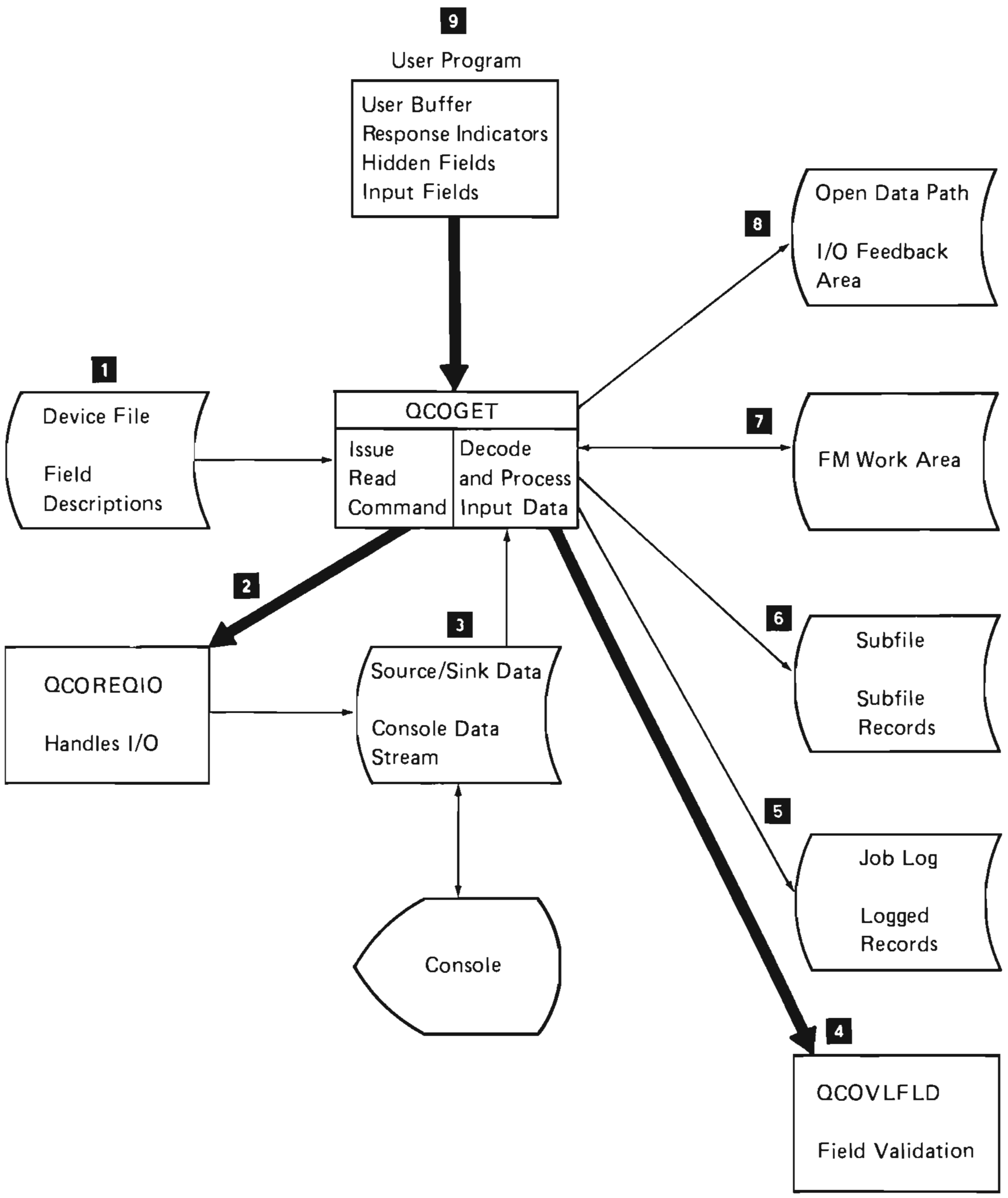


Figure CO-6. Get Operation

Get From Subfile Record (Data Flow)

Figure CO-7 shows an overview of the data flow for a get from subfile record operation.

1 Actual data does not come from the device file during a get from subfile operation.

2 Input records can optionally be logged. The data is taken from the user buffer, after being put there by QSFGET, and sent to the job log.

3 A get from subfile operation does not cause an I/O console operation. Instead, control is transferred to QSFGET to retrieve the requested record from the subfile.

4 QSFGET locates the correct record and moves it to the user buffer. All field processing and validating was done when the data was received from the console (see Figure CO-6). Because the records are stored in the subfile with the controls to redisplay them, the fields are placed in the user buffer one field at a time.

5 Output only fields are also returned to the user buffer when the record is a subfile record.

6 The I/O feedback area is updated at various points by the function manager while performing the get operation. In addition to the normal information, certain subfile information, such as relative record number returned, is also included in the I/O feedback area.

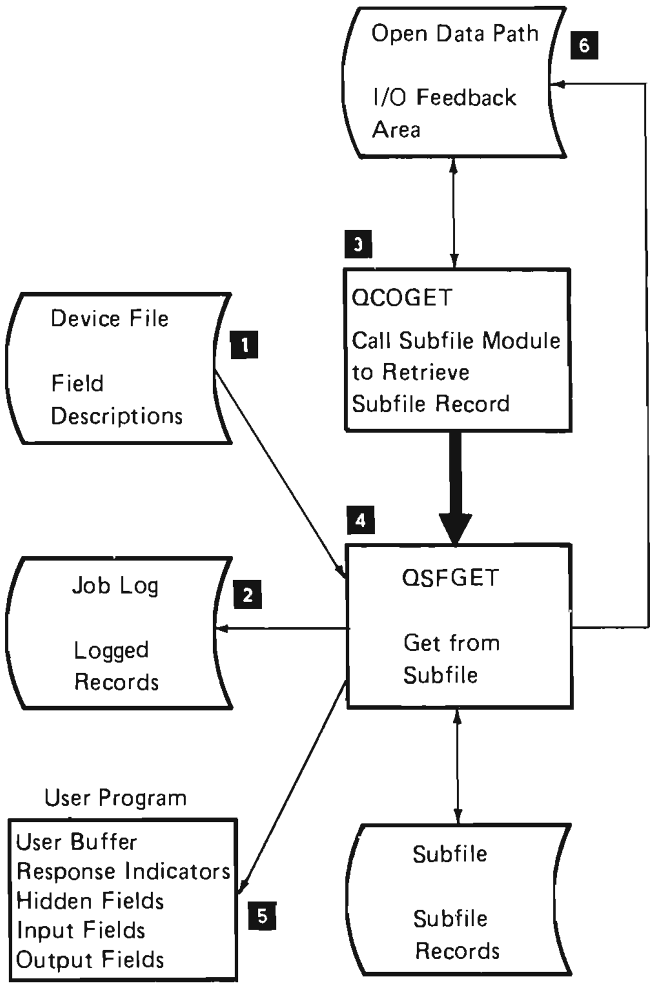


Figure CO-7. Get from Subfile Record (Data Flow)

This page is intentionally left blank.

Pass Option of the Suspend Module

The pass option of QCOSPEND indicates that the console user intends to pass across processes this display device and the unformatted data that was received on the last input request.

QCOSPEND places a pointer to the last request block used by the get operation. A pointer to this new request block is placed in the LUD (logical unit description) associated space. This is how the actual request block is passed to the new process. By passing the logical unit description lock, the new process has addressability to the request block, which contains the unformatted data that was last read. QCOCLOSE always checks to see if a passed request block exists and, if one is found, destroys it along with the regular file request block.

Get (or Put-get) Nowait Function

The get (or put-get) nowait request involves functions by several console function manager modules. Figure CO-8 and the following text describe the get (or put-get) nowait function.

1 The using program requests a get (or put-get) nowait operation.

2 QCOGET (or QCOPUT) processes the user request similar to a wait request, except that, before calling QCOREQIO to do the request I/Os, it indicates in the request block that this a nowait request. When QCOREQIO returns, QCOGET (or QCOPUT) returns to its caller.

3 QCOREQIO recognizes the nowait request and performs the REQIO instruction but does not wait for the request to complete nor does it do the dequeue of that request. Instead, QCOREQIO returns to its caller.

4 The machine issues the request I/O to the display as usual.

5 Up to this step, each module has processed the user request and returned to its caller. No module has waited for a response from the console device. The invocation stack contains only the using program.

6 When the operator responds to the get or the console device acknowledges the put, the machine enqueues the corresponding request block on the machine interface response queue in the normal manner. Because the nowait flag is set on in the request block, the machine signals the REQIO complete event.

7 The REQIO complete event invokes the nowait event handler module, QCOMEEH, which does a dequeue of the completed request block.

8 QCOMEEH calls QCOREQIO to process the dequeued request block.

9 If the request was a get nowait, QCOMEEH calls QCOGET to process the input data. QCOGET recognizes the event handler call and processes just the user input data. If the operator requested a function-manager function, such as print or help text, QCOGET handles the request and reissues the nowait request.

10 When control is returned to QCOMEEH from QCOREQIO, QCOMEEH checks to see if valid data was entered. If valid data was entered, the data available event is signaled to the user. In an operator requested function, the nowait event handler module will not see any valid operator data and QCOMEEH terminates without signaling the data available event (flow returns to **3**).

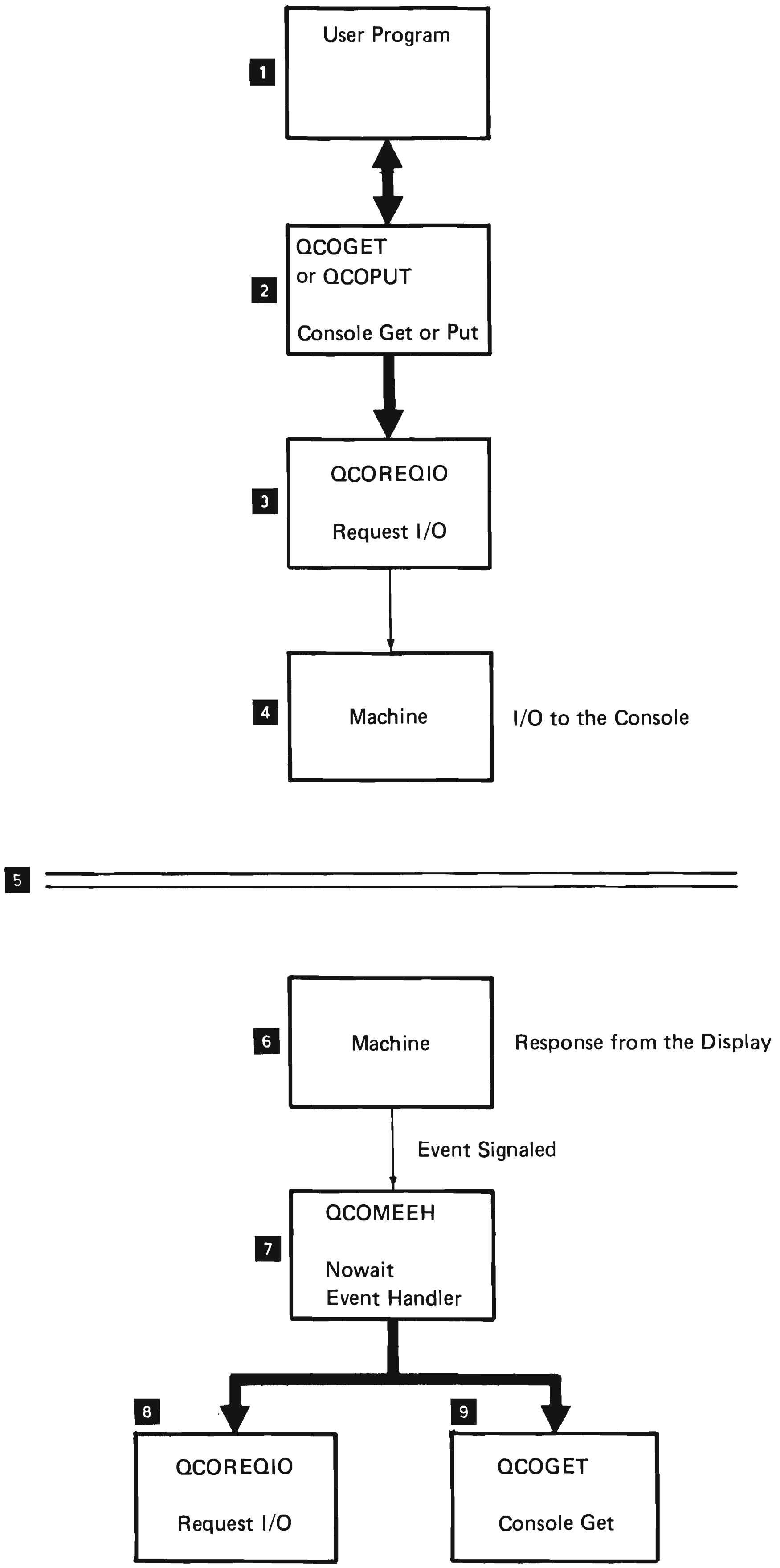
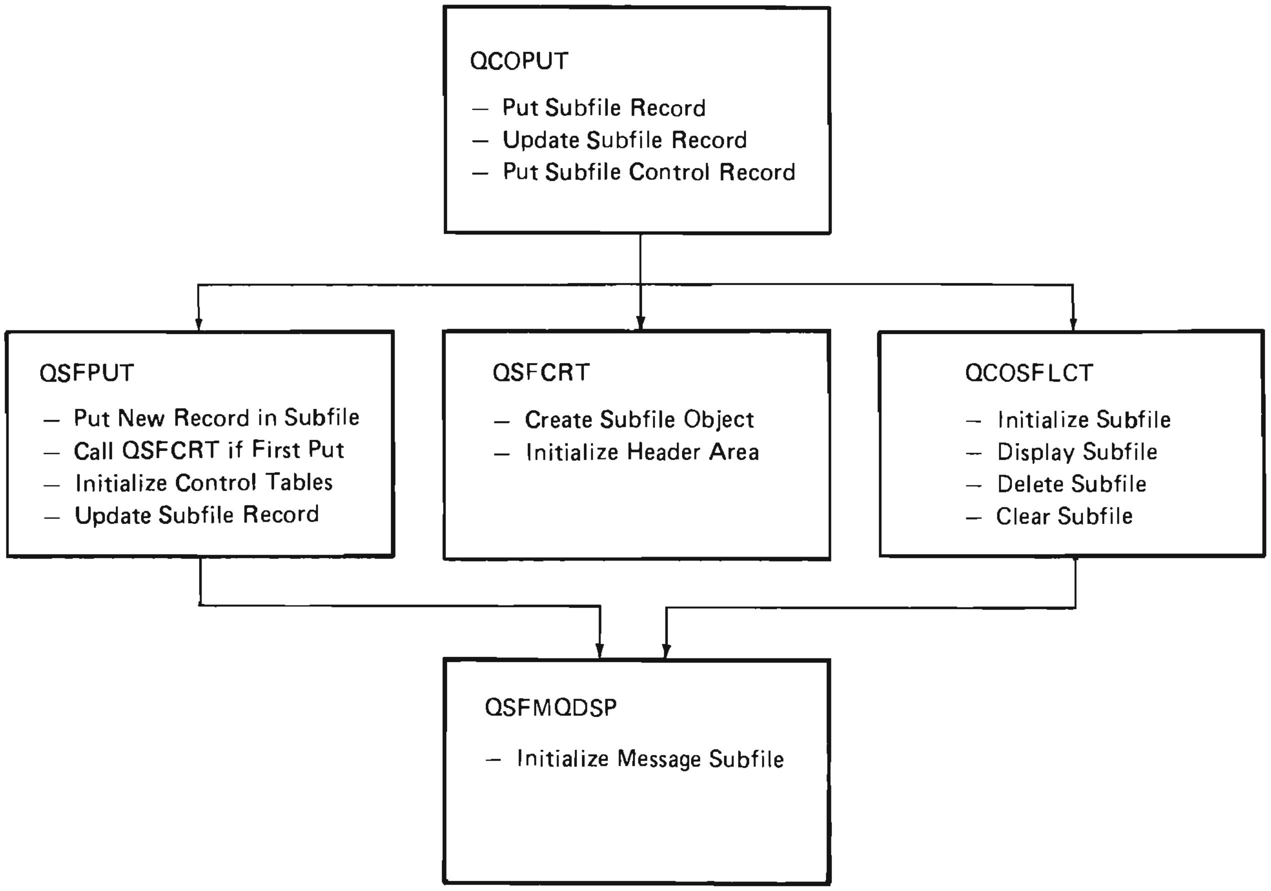


Figure CO-8. Get (or Put-get) Nowait Function

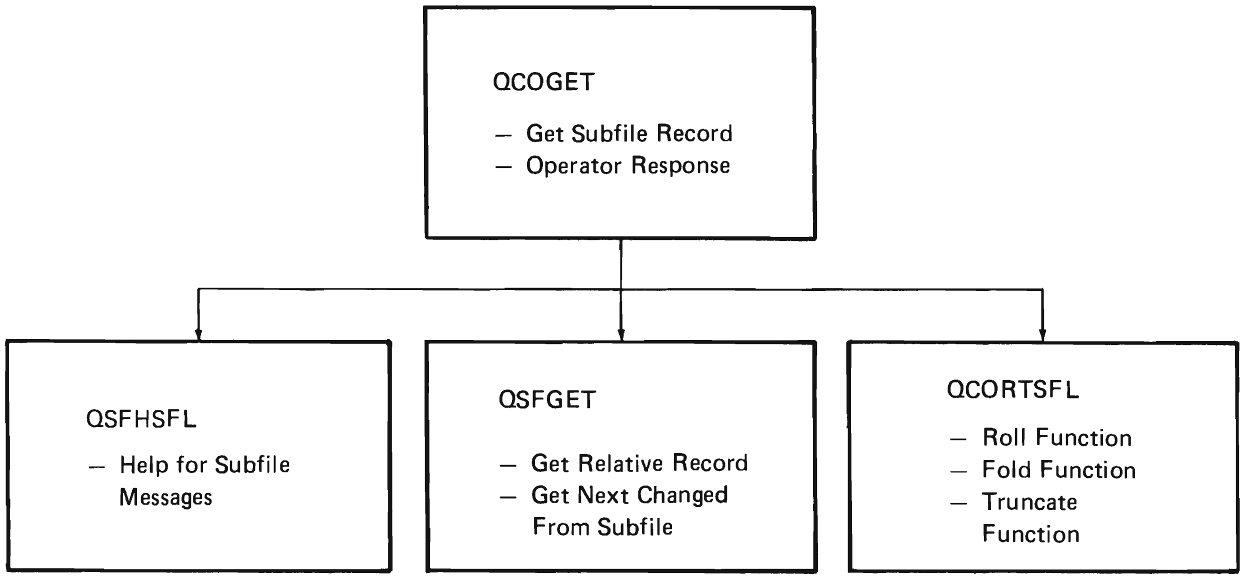
Subfile Record (Module Flow)

Figures CO-9 and CO-10 show the module flow for a put to subfile and a get from subfile record.



Notes:
1. QSFPUT is invoked only for operations directed at the subfile record.
2. QCOSFLCT is invoked only for control record operations.

Figure CO-9. Put to Subfile Record (Module Flow)



Notes:
1. QSFHSFL is called only if the operator puts the cursor in a displayed subfile of messages and presses the Help key.
2. QCORTSFL is called only if a subfile is displayed and the CA/CF key is pressed for fold/truncate or roll.

Figure CO-10. Get from Subfile Record (Module Flow)

INTRODUCTION

The copy component of the CPF (control program facility) provides the copy file function. This function is used to copy all or part of a device file, spool inline, or data base file to another device file or data base file. Records to be copied can optionally be selected on the basis of data content, relative record number, or key. If a source file is being copied, new sequence numbers, zero dates, or both can be inserted in the source fields. As part of the copy file function, the records being copied, the records being excluded, or both can be printed in either a hexadecimal and character format or in character-only format.

GENERAL OVERVIEW

Copy Modules

The copy component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QCPEXOFL–Copy File (CPYF,CPYFRMDKT, CPYTODKT, CPYFRMTAP, CPYTOTAP, CPYSRCF)[1]: This module error checks user input and calls other copy modules to complete the copy file function.

QCPFLD–Field-Level Processor: This module is called if both the from-file and the to-file are data base files with formats that are not identical. It compares the difference between record formats and the FMTOPT parameter option specified, and sends diagnostic and escape messages as appropriate. It builds a field mapping table containing attribute information for each field to be mapped from the input buffer to the output buffer. QCPFLD also validates the selection parameters specified, and sets fields in the copy control block based on user-defined record selection values for the FROMKEY, TOKEY, INCCHAR, and INCREL parameters. If INCREL is specified, a record selection table is built.

[1]This module is a CPP (command processing program).

QCPEXCON–Copy File Execution: This module opens the files, calls a module to perform the I/O operations, and closes the files. QCPEXCON uses the copy control block, and other tables to control its processing. QCPEXCON uses the following modules:

QCPFRMBR–Process Members and Labels: This module is called if the from-file and the to-file are both data base files, or if a multiple data base member or diskette label copy is requested. It builds a member table containing a list of from-file members or labels to be copied.

QCPCREAT–Create a To-File and Members: If the from-file is a data base file, CRTFILE (*YES) is specified and the to-file does not exist, this module is called to create a physical file and members with the same record format and access path as the from-file. If the to-file exists, but the member(s) copied do not, then QCPCREAT is called to add the necessary members.

QDBFFCPY–Fast Copy: This module performs fast physical data base file to physical data base file I/O, when individual records do not have to be processed.

QCPGENIO–General I/O Routine: This module performs blocked record I/O for all copy functions.

The following module is used by QCPEXCON and QCPGENIO.

QCPPRINT–Copy File Print: This module is called when TOFILE (*LIST) or the PRINT parameter is specified, to print headings and messages on the listings.

QCPCLNUP–Copy File Clean-Up: This module gets control from QCPEXOFL for normal copy completion, or by machine functions as an invocation exit for an error termination. QCPCLNUP closes files (if necessary), releases file locks, and destroys the temporary spaces created by copy for use as workareas.

Copy File Operation Overview

Figure CP-1 and the following text show the copy file operation.

1 The copy file function may be invoked by CL input of the Copy File (CPYF), Copy From Diskette (CPYFRMDKT), Copy to Diskette (CPYTODKT), Copy From Tape (CPYFRMTAP), Copy to Tape (CPYTOTAP), and Copy Source File (CPYSRCF) commands.

The copy file function may also be invoked by another CPF component using a macro interface. The macro interface offers only a limited subset of the parameter options that are available to the CL user.

Each command is syntax checked by the command analyzer before QCPEXOFL is passed the command parameters. QCPEXOFL performs validity checking and preexecution setup, then calls an execution module. Record selection and copying, and the print function are performed based on the file characteristics and user options specified.

2 Based on information about the from-file and to-file, QCPEXOFL performs command validity checks to detect errors that were not discovered by command analyzer syntax checking. The following types of errors can be detected:

- Library, file, or format cannot be found.
- From-file or to-file is not a valid type.
- User not properly authorized.
- Parameters supplied were not appropriate for the file characteristics.
- Parameters required by the file characteristics were not supplied.
- Conflicting parameters were entered.

The from-file and to-file are locked and various file extracts are done to determine file attributes. If an error is detected, the copy component terminates the command and issues the appropriate exception message. If the command is valid, QCPEXOFL calls another module to do further preexecution setup and then calls the execution module. A copy work space and two temporary spaces (for use by file and member extracts) are created by QCPEXOFL, updated by QCPEXOFL and other modules. The copy work space contains the copy control block, and all other control blocks and tables built by copy to control execution. A pointer to the copy control block is passed to the execution module for use in controlling the copy operation. Pointers to the tables created are stored in the copy control block.

QCPEXOFL transfers control to QCPCLNUP, and QCPCLNUP performs normal copy completion. QCPCLNUP may also be invoked by machine function as an invocation exit.

3 If the from-file and the to-file are both data base files with nonidentical record formats, QCPFLD will build a field mapping table. This table indicates how fields are mapped (moved and converted).

QCPFLD is also called to validity check and set fields in the copy control block, based on the selection parameters entered on the copy command. If INCREL is specified, a field selection table is built for use during the copy execution phase.

4 QCPEXCON controls the actual data copy. The from-file can be a diskette, card reader, tape, spool, inline file, physical file, or logical file. The to-file can be a diskette, card punch, tape, printer, physical file, or special value *LIST.

QCPEXCON performs the file open and close functions. Get, record selection, field mapping, and put operations are performed by QCPGENIO or QDBFFCPY. A pointer to the copy control block is passed as a parameter for use in controlling I/O operations.

5 If the from-file and the to-file are both data base files, or if a multiple data base member or diskette label copy is requested, QCPFRMBR is called to extract member information. A member table is built that contains the list of from-file members or diskette labels to be copied.

6 If the to-file does not exist, QCPCREAT will create a physical file and members with the same record format and access path as the from-file, provided CRTFILE (*YES) has been specified. If the to-file exists, QCPCREAT will create any member in the to-file that does not exist, using the name of a corresponding member in the from-file.

7 QCPGENIO performs I/O operations for all types of files, including printing record data on the *LIST, *COPIED, and EXCLD listings. Records are printed in hexadecimal and character format or character-only format, 100 characters to a print line.

8 Print Function: QCPPRINT is called by QCPEXCON and QCPGENIO to print headings and messages on the *LIST, *COPIED, and *EXCLD listings. A listing block is allocated and initialized by QCPPRINT, and used by QCPPRINT and QCPGENIO.

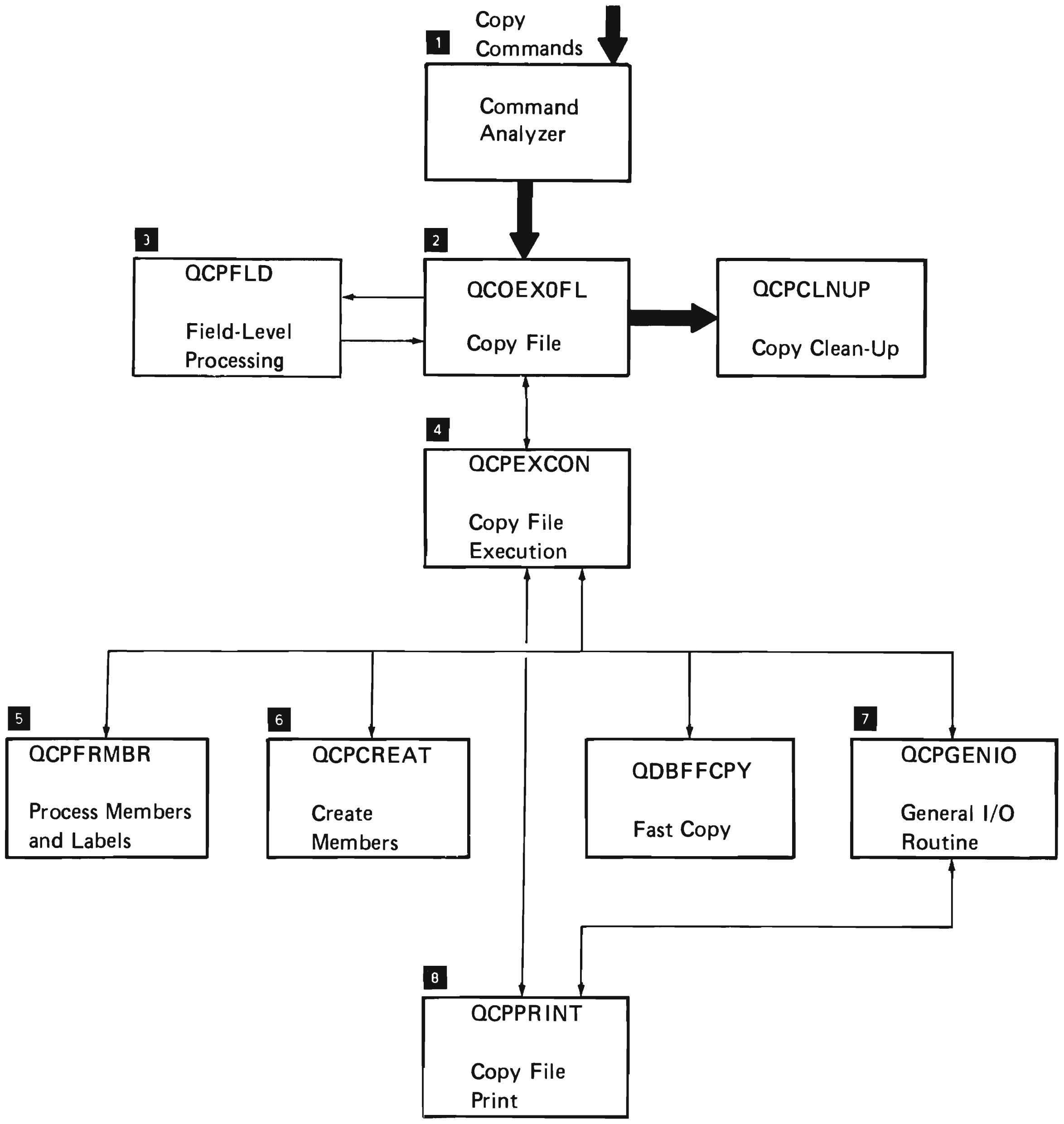


Figure CP-1. Copy File Operation Overview

INTRODUCTION

The 5424 function manager component of the CPF (control program facility) provides the support for the 5424 MFCU (multifunction card unit) on System/38.

The following MFCU functions are supported by the 5424 function manager:

- Open MFCU file for processing
- Close MFCU file to processing
- Read data from an MFCU input file
- Write data to an MFCU output file
- Read and write data to MFCU I/O files

GENERAL OVERVIEW

5424 Function Manager Modules

The 5424 function manager consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QCSCLOSE–Card Close: This module closes a file to processing on the 5424 MFCU.

-->QCSGET–Card Get: This module reads records from the 5424 MFCU.

-->QCSOPEN–Card Open: This module opens a file to processing by the 5424 MFCU.

-->QCSPTGT–Card Put/Get: This module performs both put and get operations for I/O operations on the 5424 MFCU.

-->QCSPUT–Card Put: This module sends records to the 5424 MFCU to be punched and/or printed on the cards.

-->QCSXGERR–Exception Generator and Error Handler: This module handles all exceptions and generates all CPF messages.

-->QCSFEOD–Forced End-of-Data: This module forces the write of internally-buffered data or read to an EOF end-of-file).

-->QCSEVT–Event Handler: This module handles the operator intervention required event.

-->QCSLUDIN–Card LUD Initialization: This module initializes the 5424 MFCU LUD (logical unit description) associated space.

5424 Function Manager Operation

Figure CS-1 and the following text describes the operation of the 5424 function manager.

1 A high-level language program or the spooling component, through the QDMCOPEN module of common data management, calls QCSOPEN to open a card device file for I/O processing.

- **A** An argument list is passed that contains a pointer to the UFCB (user file control block) and an index into the ODPCB (open data path control block) for the device being opened.

- **B** Record lengths and the hopper to be used for the I/O operations are determined and appropriate messages or exceptions are generated.

The type of operation to be performed on the 5424 MFCU is determined and the necessary objects for that operation are created and initialized.

- **C** Two request I/Os are issued to fill the two buffers used when nondevice-dependent input operations are requested, one REQIO is needed to feed a card from the specified hopper to the wait station when output operations are requested, and one REQIO is needed to read a card for input on all combined files.

2 After a file has been opened for input or I/O operations, the interface to QCSGET is valid.

- **A** An argument list is passed that contains a pointer to the UFCB, a pointer to a control list, and a pointer to an option list.

- **B** If the option list does *not* specify option wait, an exception is signaled.

A card is read from the MFCU and the data is returned to the user for I/O files and device-dependent input operations. For nondevice-dependent input operations, records are returned, one at a time, for each get from the buffers that were filled by the open operation.

- **C** When a buffer is empty, cards are read to refill it. If an I/O error is detected, error recovery is attempted.

- **D** When EOF (end-of-file) is detected, an exception is signaled.

3 When a file has been opened for output or I/O operations, the interface to QCSPUT is valid.

- **A** An argument list is passed that contains a pointer to the UFCB, a pointer to a control list, and a pointer to an option list.

- **B** If the option list does *not* specify option wait, an exception is signaled.

- **C** An I/O operation to punch cards, print cards, or both is performed for output device-dependent control operations. If an I/O error is detected, error recovery is attempted. For nondevice-dependent control operations, data is accepted from the user and put into a buffer. When the buffer is full, an I/O operation is performed and a buffer switch takes place. A full buffer is immediately sent to the 5424. If an I/O error is detected, error recovery is attempted.

4 When a file has been opened for I/O operations, the interface to QCSPTGT is valid.

A An argument list is passed that contains a pointer to the UFCB, a pointer to a control list, and a pointer to an option list.

B If the option list does *not* specify option wait, an exception is signaled.

C The first operation to an I/O file defaults to a read operation; subsequent operations cause first an output operation and then a read operation to occur. The output operation is performed on the card that was previously read. If an I/O error is detected, error recovery is attempted.

B When an EOF is detected, an exception is signaled.

5 The interface to QCSFEOD is valid for any type of open.

A An argument list is passed that contains a pointer to the UFCB.

C For either input or I/O operations, reads are performed until an EOF is detected. An EOF exception is then signaled. For output files, any blocked data is sent to the device.

6 When a high-level program or the spooling component is finished with a file, it closes the file by calling QCSCLOSE through the QDMCLOSE module of common data management.

A An argument list is passed that contains a pointer to the ODPCB, an index to the device being closed, and the type of close that is to be performed on the file.

C Normal and permanent type close for output files causes partially filled buffers to be sent to the MFCU. If an error is detected, error recovery is attempted. The objects created by the open to support the output functions are destroyed. Control is returned to QDMCLOSE.

Normal and permanent type close for input files causes any requests for input data sent to the I/O manager to be recalled. The objects created by the open to support the input functions are destroyed. Control is returned to QDMCLOSE.

Normal and permanent type close for I/O files causes the objects created by the open to support the I/O functions to be destroyed. Control is returned to QDMCLOSE.

Temporary type close is the same as a normal and permanent type close, except that the objects created to support the functions are not destroyed.

Abnormal type close causes all I/O activity to stop. All objects are destroyed and control is returned to QDMCLOSE.

7 C QCSEVT is called whenever an intervention required event is signaled by the 5424 I/O manager.

B A message is sent to the operator console telling the operator that the 5424 is not ready.

8 B QCSXGERR is called to send error messages and exceptions.

C When an I/O error is detected, error recovery is attempted.

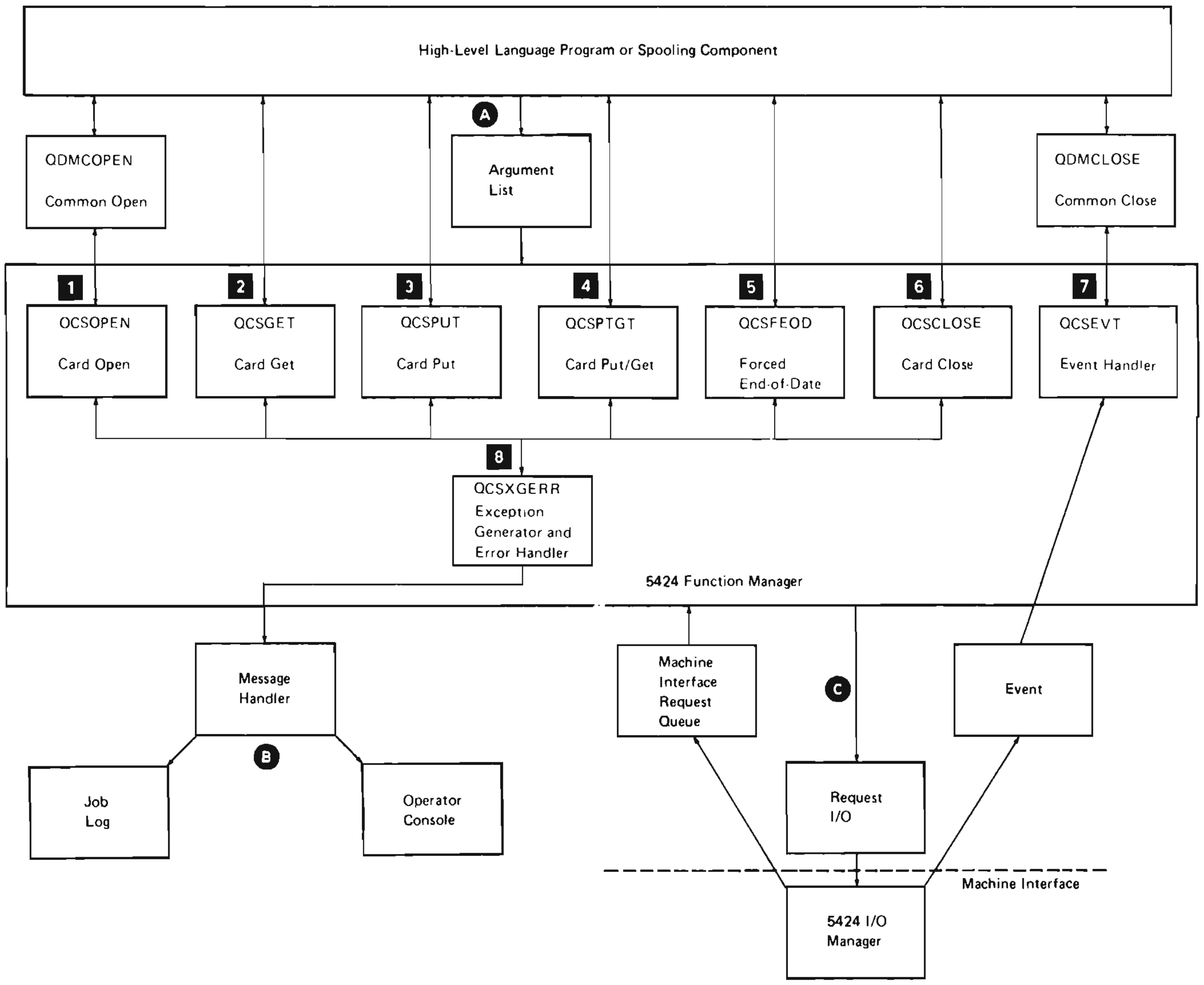


Figure CS-1. 5424 Function Manager Overview

INTRODUCTION

The data base component of the CPF (control program facility) controls the existence of data-containing files, provides information about those files, and allows access to the data in their members.

The types of data base files supported are physical, logical, logical join files, and logical derived files. Physical files can actually hold data records. Logical files and logical join files access data from one or more physical files by means of record formats, access path, or both (they are different from the physical file record formats and access path). Logical derived files access the data through another logical or a physical file's keyed access path.

The functions that make up the data base component fall into the following categories:

- Definition: The creation, change, and destruction of data base files and the addition, change, rename, and removal of members of data base files.
- Manipulation: Operations on the data in a data base file member, including open and close, copy, reorganize, get/put/update/delete/release, clearing or initializing a member, forcing end-of-data for a member, and querying.
- Extraction: Retrieval of information on the structure, interrelationship, and status of data base files and members. Check file existence and authority (for librarian), calculate file size (for librarian), dump file object (for service), display data object locks (for Display Object Lock, Display Active Job, and Display Job commands), and display record locks (for Display Record Lock, and Display Job commands).
- Data base recovery: Recovery of the data base from cancellation of jobs or from a system failure; that is, completing or backing out interrupted crash-sensitive functions, handling physical file members that have their data changed, and rebuilding access paths as necessary.
- Data base event handling: Response to system-wide events involving the data space indexes that are part of keyed data base file members and the data spaces that are part of physical data base file members.
- Data base file handling for generic operations that change an entire file: Completion of functions such as move or rename object, grant or revoke authority to an object, and change object owner.
- Data base file handling for save/restore and reclaim:
  - Data base reclaim storage: Lost data is recovered and placed in the QRCL library, and lost data base control blocks are eliminated if data is not being addressed. This function is performed during the execution of the reclaim storage facility.
  - Data base save/restore: Handles part of the processing for those objects in a save or restore request that are data base files.

Entry to Data Base Functions

All of the previously named categories, except the recovery and event handling functions are visible to users of data base files. The recovery and event handling functions of the data base component appear automatic to users of the component. The recovery function is invoked by the work control component when it is performing its start CPF function. The event handlers are invoked by monitors that listen for asynchronous system-wide events as long as CPF is running.

The user entry to most of the data base functions is through the common data management component. Figure DB-1 and the following text describe the paths of invocation of the data base modules.

1 QDBCLOSC, QDBCLRPF, QDBINZPF, QDBOPENC, QDBRNMME, QDBRGZPF, QDBCHGFI, QDBRGZPF and QDBCHGME are command processing programs.

Note: QDBCLRPF may also be invoked by QDBOPEN.

2 The open/close and I/O functions are entered through macros of the common data management component.

A There are open and close functions common to data base and device files that are performed by QDMCOPEN and QDMCLOSE before the data base open and close are invoked.

B An I/O macro determines which data base module to invoke for its function by using a table in the open members' open data path. There is an element in this table for each operation that is valid for the open member; that element is an offset into the system entry point table so that it identifies the entry that addresses the data base I/O module.

C The query functions are entered through data base macros. QDBQUERY creates the queries and the open or close is performed by existing I/O support.

3 The QDMROUTE module of common data management is the interface to the definition, extraction, and generic functions of both the data base and the device file definition components. It transfers control to a data base module after determining that the file for which it was invoked is a data base file (as opposed to a device file).

D The generic functions, such as rename or grant authority, are reached through the general object interfaces of the components responsible for the function.

The command interface to data base definition function is provided by the data definition component. The command interface to data base extraction functions is provided by the file reference function component.

E The data base reclaim function is reached through the reclaim storage facility.

The data base check object function is reached through the librarian component Check Object (CHKOBJ) command.

The data base save/restore function is reached through the Save Object (SAVOBJ), Save Changed Object (SAVCHGOBJ), Save Library (SAVLIB), Restore Object (RSTOBJ), Restore Library (RSTLIB), and Save System (SAVSYS) commands.

The data base display lock function is reached through the Display Object Lock (DSPOBJLCK), Display Active Job (DSPACTJOB), and Display Job (DSPJOB) commands.

The data base display record lock is reached through the Display Record Lock (DSPRCDLCK) and Display Job (DSPJOB) commands.

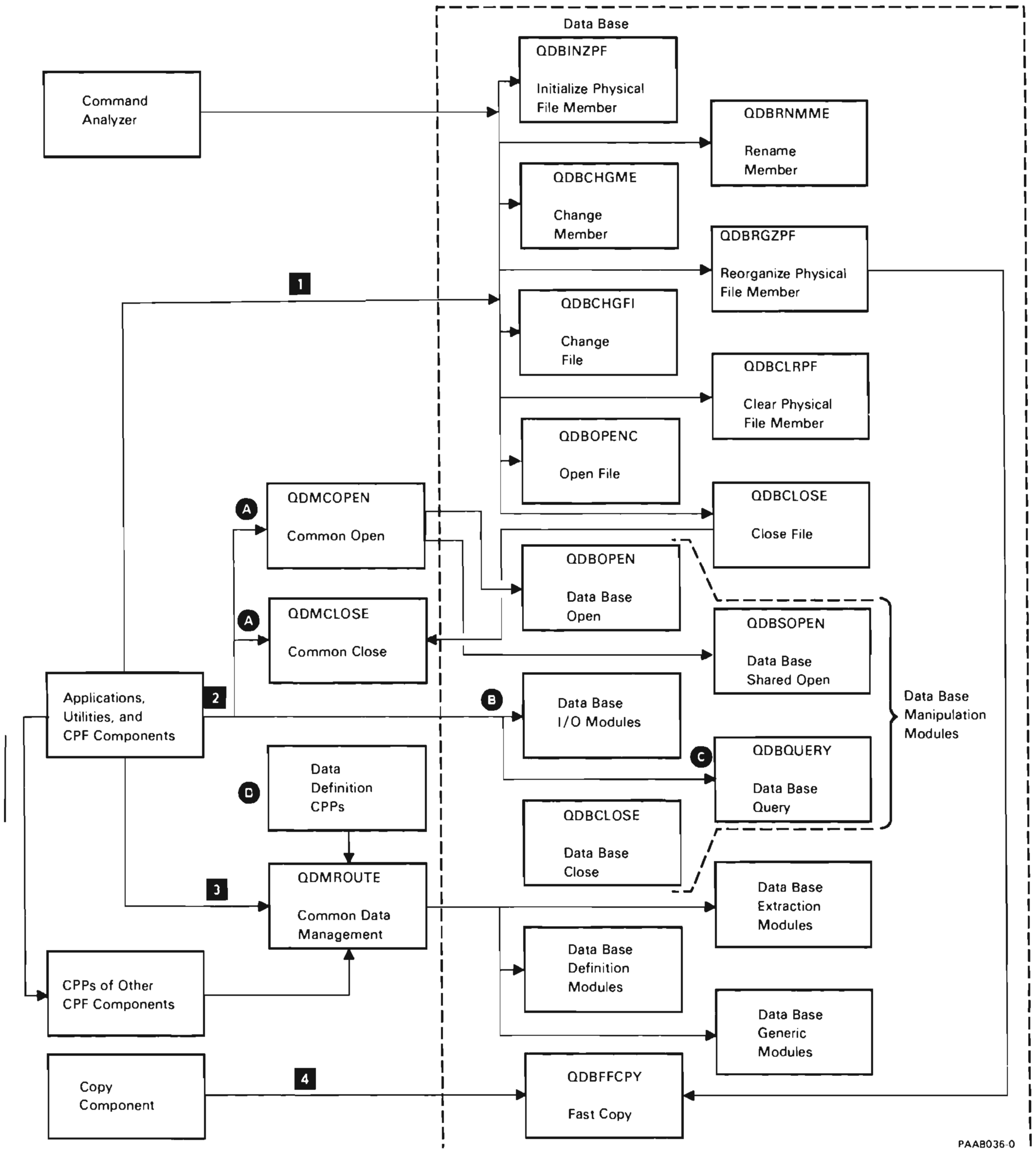
The data base create duplicate file function is reached through the librarian component Create Duplicate Object (CRTDUPOBJ) command.

The data base convert file function is reached through the install component or data base reclaim function.

The data base dump file function is reached through the service component.

The data base size file function is reached through the librarian component.

4 Copy is a manipulation function. The data base fast copy and reorganize function is reached through the copy component and the data base module QDBRGZPF.



Data Base
Command Analyzer
QDBINZPF Initialize Physical File Member
QDBRNMME Rename Member
QDBCHGME Change Member
QDBRGZPF Reorganize Physical File Member
1
QDBCHGFI Change File
QDBCLRPF Clear Physical File Member
QDBOPENC Open File
A
QDMCOPEN Common Open
QDBCLOSE Close File
QDBOPEN Data Base Open
A
QDMCLOSE Common Close
QDBSOPEN Data Base Shared Open
Applications, Utilities, and CPF Components
2
B
Data Base I/O Modules
Data Base Manipulation Modules
C
QDBQUERY Data Base Query
D
Data Definition CPPs
QDBCLOSE Data Base Close
3
QDMROUTE Common Data Management
Data Base Extraction Modules
Data Base Definition Modules
CPPs of Other CPF Components
Data Base Generic Modules
Copy Component
4
QDBFFCPY Fast Copy
PAAB036-0

Figure DB-1 (Part 1 of 2). Entry to Data Base

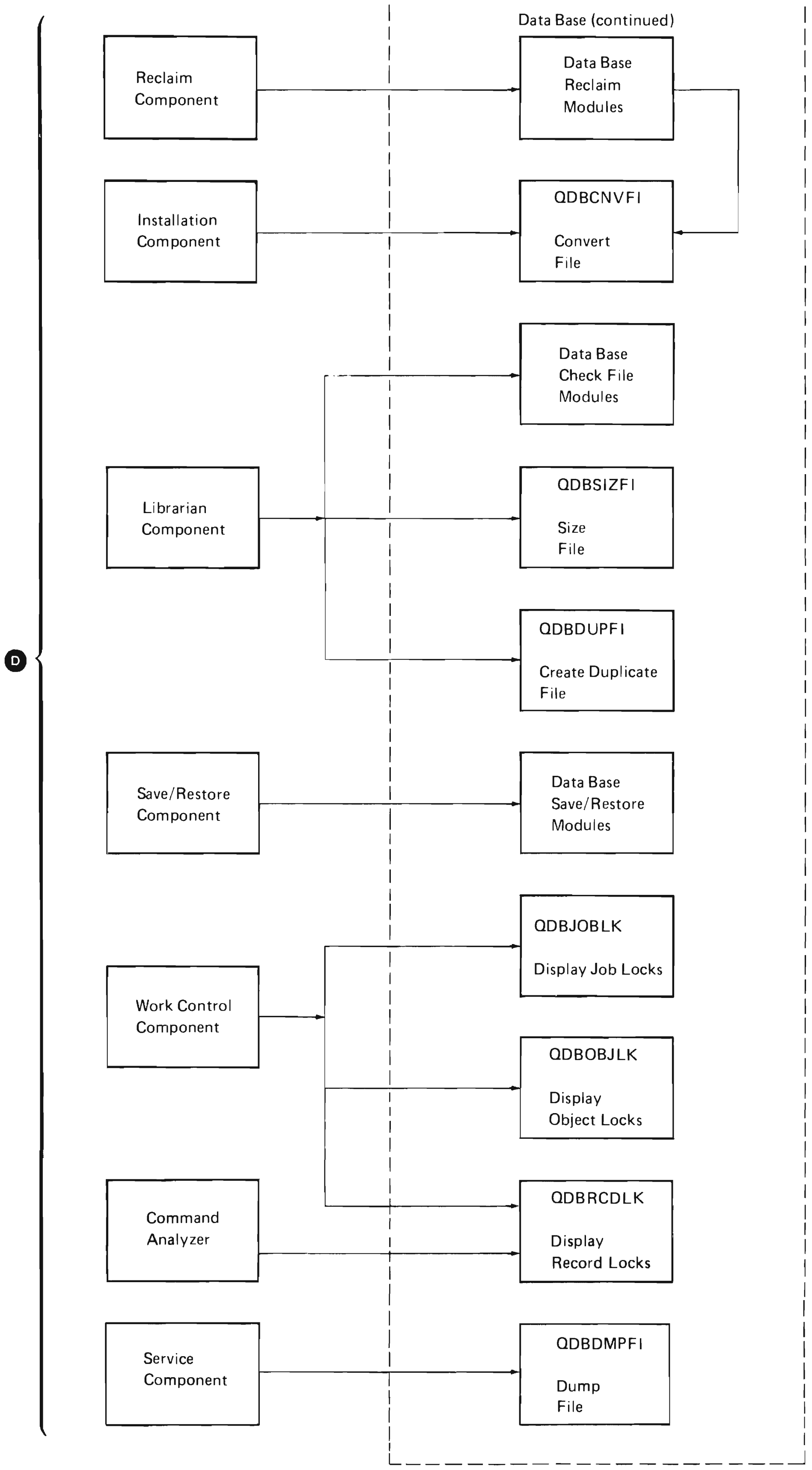


Figure DB-1 (Part 2 of 2). Entry to Data Base

GENERAL OVERVIEW

Data Base Modules

The data base components' functions are performed by the following modules. They are grouped by the previously mentioned functional categories.

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

Data Base Definition Modules

-->QDBCRTFI–Create Data Base File: This module creates a new data base file with no members.

> QDBCRTFS–Create File Select/Omit Processing: This module processes the select/omit specification, if any, for the creation of a logical file with select/omit.

-->QDBCRTME–Create Data Base File Member: This module creates a new member in a data base file.

> QDBCRTMO–Create Member Ownership: This module changes ownership of the new member to the file owner.

-->QDBDLTFI–Delete Data Base File: This module destroys a data base file, including all its members. If damage to the file or its members is detected, a message is sent and the deletion continues.

-->QDBDLTME–Delete Data Base File Member: This module removes a member from a data base file. If damage to the members is detected, a message is sent and the deletion continues.

The following module is used by QDBCHGFI, QDBDLTFI, QDBCRTME, and QDBDLTME.

> QDBISHRX–Implicit Access Path Sharing: This module does all the processing necessary for implicitly shared access paths.

The following module is used by QDBCRTFI, QDBCRTME, QDBDLTFI, QDBDLTME, and QDBISHRX.

> QDBDIRUP–Update Data Base Directory: This module adopts a user profile to add/remove a file or member from a data base directory.

-->QDBCHGFI: Change Data Base File: This module changes the attributes of a data base file.

-->QDBCHGME: Change Data Base Member: This module changes the attributes of a data base member.

-->QDBRNMME–Rename Data Base Member (RNMM)[1]: This module renames a data base member.

-->QDBDUPFI–Create Duplicate Data Base File: This module creates a data base file which is a duplicate of another data base file.

Data Base Manipulation Modules

-->QDBOPEN–Data Base Open: This module is called by QDMCOPEN to complete the setup of a data base file member so that its data can be accessed by a program. It performs checks such as expiration date, initializes sections of the ODP (open data path), and activates cursor copy of the member. If the member's data space index is invalid (in a nonmaintained state) and is to be used for this open, it is rebuilt. If the member is to be cleared, QDBCLRPF is called to clear the member.

-->QDBOPENC–Data Base Open CPP (OPNDBF)[1]: This module invokes the common data management open function to open a data base file (results in call to QDBOPEN).

-->QDBSOPEN–Data Base Shared Open: This module performs open option consistency checking on a shared data base open.

-->QDBGETSQ–Data Base Get Sequential: This module performs the get of a record from a data base file member for the options of FIRST, LAST, NEXT, PREVIOUS, and SAME.

-->QDBGETM–Data Base Get Sequential Multiple: This module performs the get of a group of records from a data base file member for the option NEXT.

-->QDBGETDR–Data Base Get Direct: This module performs the get of a record from a data base file member for the relative record options.

[1]This module is a CPP (command processing program).

-->QDBGETKY–Data Base Get by Key: This module performs the get of a record from a data base file member for the options that specify a key and for get next/previous unique.

-->QDBSEQMP–Data Base Sequential Member Processing: This module opens the next member if OVRDBF MBR (*ALL) has been specified.

-->QDBPUT–Data Base Put: This module inserts a record into a data base file member and handles force write, inhibit write and file increments.

-->QDBPUTM–Data Base Put Multiple: This module inserts a group of records into a data base file member and handles force write, inhibit write, and file increments.

QDBPUTMX–Data Base Put Multiple Exception Handler: This module forces all records in the group, up to the record in error, and then signals the exception to the user.

-->QDBUDR–Data Base Update/Delete/Release: This module performs the update, delete, or release of a record that was locked for update by a get against the same data base file member.

The following module is used by QDBOPEN, QDBGETSQ, QDBGETDR, QDBGETKY, QDBPUT, QDBUDR, QDBGETM, QDBPUTM, and QDBPUTMX.

QDBSIGEX–Data Base I/O Signal Exception: This module signals any status, notify, or escape message from open and I/O operations. It also handles the response to a notify message.

-->QDBFEOD–Data Base Force End of Data: This module sends an end of file message and forces changes in a member to secondary storage. If there are any records in a SEQONLY (*Yes) output buffer, they are added to the member and forced.

-->QDBCLOSE–Data Base Close: This module permanently sets an open data base file member to a state that prevents the program from accessing its data.

-->QDBCLOSC–Data Base Close CPP (CLOF)[1]: This module invokes the common data management close function to close a data base file (results in a call to QDBCLOSE).

-->QDBQUERY–Data Base Query: This module creates a query member for use by I/O modules.

QDBEXIT–Data Base Exit: This module handles cleanup during invocation cancelation.

Data Base Member Modules

-->QDBINZPF–Data Base Initialize Physical File Member (INZPFM)[1]: This module is used to add either default or deleted records to a member of a physical file, placing them after existing records in the member.

-->QDBCLRPF–Clear Physical File Member (CLRPFM)[1]: This module is used to empty a physical file member of records.

-->QDBRGZPF–Reorganize Physical File Member (RGZPFM)[1]: This module performs the entire reorganize physical file function, including the removal of deleted records, the resequencing of records by key, and the updating of the source file sequence number and date fields.

QDBFFCPY–Fast Copy: This module performs the compress and optional reorganize function.

Data Base Extraction Modules

-->QDBEXDFI–Extract from Data Base File: This module provides the list of a file's members, the definition of a file, a format, a field in a format, a list of a file's formats, or the definition of a file's keys.

-->QDBEXDME–Extract Data Base File Member: This module provides the definition of a member of a data base file and such status information as its size and deleted record count.

-->QDBEXTWU–Extract Data Base Where-Used: This module provides either a list of files using a format, a list of files sharing the data or access path of a file, or a list of members sharing the data or access path of a member.

QDBEXTEX–Data Base Extract Invocation Exit Program: This module handles a cancel request and unlocks locked files and members.

-->QDBDMPFI–Dump Data Base File: This module dumps the constituent objects of a data base file and its members.

[1]This module is a CPP (command processing program).

-->QDBSIZFI–Data Base File Size: This module calculates the size of a data base file, including its members.

-->QDBCHKFI–Check Existence and Authority to Data Base File: This module checks the existence of and authorization to a data base file and/or member.

-->QDBOBJLK–Display Object Locks: This module is invoked to extract information on locks held on parts of a data base file.

-->QDBJOBLK–Display Job Locks: This module is invoked to extract information on locks held by the job on parts of a data base file.

-->QDBRCDLK–Display Record Locks: This module is invoked to extract information on locks held on data base records.

Data Base Recovery Modules

-->QDBRCIPS–Data Base Synchronous Recovery: This module runs during the start CPF process. It invokes a recovery function for each create, change, delete, move, rename, grant, revoke, transfer, or restore that was in process at crash. It also displays all indexes requiring recovery at IMPL and the recovery options. The user may override the displayed recovery option for this IMPL. It rebuilds indexes which were defined as synchronous recovery indexes, and locks other immediate or delayed maintenance indexes. It sends messages about damage to objects that constitute data base files.

QDBCDFIR–Data Base File Create Recovery: This module recovers from an interrupted create file operation by deleting all pieces of the file. It is called either by QDBRCIPS, by exception handling in QDBCRTFI, or by QDBFIXIT.

QDBCDMER–Data Base File Member Create/Delete Recovery: This module recovers from an interrupted create or delete member operation by deleting all pieces of the member. It is called either by QDBRCIPS, by exception handling in QDBCRTME, or by QDBFIXIT.

QDBMVRFR–Data Base Move/Rename File Recovery: This module recovers from an interrupted move or rename file operation by completing it. It is called either by QDBRCIPS or QDBFIXIT.

QDBAUTFR–Data Base Authorization Recovery: This module recovers from an interrupted grant or revoke of authority to a data base file or transfer of ownership of a data base file. It recovers the function by completing it. It is called either by QDBRCIPS or QDBFIXIT.

QDBRSRCV–Data Base Restore Mending: This module ensures the linkage among internal objects for data base files, which are involved in interrupted restore operations. It is called either by QDBRCIPS or QDBFIXIT.

QDBCHGFR–Data Base Change File Recovery: This module recovers from an interrupted file change operation by attempting to complete the change. It is called either by QDBRCIPS or QDBFIXIT.

-->QDBRCDYN–Data Base Asynchronous Recovery: This module rebuilds the indexes locked by QDBRCIPS.

-->QDBFIXIT–Data Base Logical Damage Recovery: This module recovers online from any interrupted file-level operation.

Data Base Event Handling Modules

-->QDBIVLIX–Data Base Invalid Index Event Handler: This module handles the invalid index event by rebuilding the index (if it was defined as immediate or delayed maintenance) and sending a message.

-->QDBCMPTH–Data Base Compression Threshold Event Handler: This module handles the data space compression threshold exceeded event by sending a message to the system operator.

Data Base Generic File Handling Modules

-->QDBMOVFI–Move Data Base File: This module transfers addressability of a data base file and its members from one library to another.

-->QDBRNMFI–Rename Data Base File: This module changes the name of a data base file.

-->QDBGRTFI–Grant Authority to Data Base File: This module grants to a user some authority to a data base file and its members.

-->QDBRVKFI–Revoke Authority to Data Base File: This module revokes from a user some authority to a data base file and its members.

-->QDBXFRFI–Transfer Ownership of Data Base File: This module transfers the ownership of a data base file and its members from one user to another.

-->QDBCNVFI–Convert Data Base File: This module converts a data base file to the current release/modification level of CPF.

Data Base Save/Restore and Reclaim Generic File Handling Modules

-->QDBRCLMA–Data Base Reclaim Storage: This module finds all pieces of a data base file, and ensures that all have consistent authorizations, ownership, names, and addressability.

-->QDBRCLMB–Data Base Reclaim Lost Cursors, Indexes, and Data Spaces: This module reclaims the storage for cursors and indexes that no longer are part of a data base file. If lost data is found, a data base file is created into the QRCL library to enable the user to recover the data.

-->QDBRCLMC–Data Base Reclaim Lost Directories and Formats: This module reclaims the storage previously used for directories or formats that no longer are part of a data base file.

-->QDBSVPRE–Data Base Save Predump Processing: This module extracts file, format, and member definitions to be saved, and lists the machine instruction objects for data and access paths to be dumped. It handles all the data base files in a save request, sorting them so they appear on the media in an order that, if used to re-create the objects, satisfies all dependencies (that is, logical or physical) between the files.

-->QDBSVPST–Data Base Save Postdump Processing: This module cleans up after the objects have been dumped.

-->QDBRSPRE–Data Base Restore Preload Processing: This module is invoked once for all files (in a library) in a restore request before loading objects. It selects members to load, creates files and/or members if they do not exist, restores authorities, and lists machine objects to load.

-->QDBRSPST–Data Base Restore Postload Processing: This module is invoked once for all files (in a library) in a restore request after loading objects in order to complete restore processing.

The Structure of Data Base Files

Figure DB-2 and the following text describe the structure of data base files. In the figure, the dotted outlines represent the composite objects that are visible to users of the data base component: files and members. The solid outlines are the machine interface objects that implement the data base. Within them, the unshaded areas contain pointers relating the machine interface objects (as represented by arrows in the figure).

In general, a data base file consists of a file definition plus its members. Data base design objects (formats and directories) implement sharing between files. The objects that compose files and members and the contents of their control blocks vary depending on whether a file is keyed or arrival sequence, and on whether it is physical, logical, or derived. In Figure DB-2, there are three files.

A) File A is a physical, unkeyed file with one member.

B) File B is a keyed logical file based on three physcial files; one of its based-on files is File A, which it views through the same format. It is shown with one of its members, which is based on the single member of file A and two other physical members.

C) File C is a derived file that shares the access path of File B but views records from the based-on physical files of File B (its parent) through three new formats. File C is shown with two members; each of them has as its parent member a member of file B.

D) File D is a keyed join logical file that implicitly shares an access path for one of its secondary indexes (second member of file B). File D is based on three physical files that it views through its format.

1 The FCB (file control block) contains several segments of information. The primary portion is the file description template (WWDBFDT include), which holds such information as file type, key definitions and all linkage pointers. The FCB of a logical file contains a scope list: an array that addresses the based-on physical files and relates each of them to a format that describes the logical view of its records. In the FCB of a logical derived file, the parent file (the file whose access path is being shared) is identified. The scope list entries point to the formats used for records from the corresponding based-on files of the parent file.

The remainder of the FCB consists of templates for the machine interface objects that constitute the members of the file. These templates are derived from information in the file and format descriptions, and are used at member creation time.

2 The format object contains a record format definition (WWDDFMTD include) consisting of a series of field descriptions. A format can be used by one or more data base files and can serve as both a physical and a logical record format. It can be addressed only through the FCB scope list of a file that uses it.

3 The machine interface object that is addressed when a data base file member is addressed is the cursor. It is through an activated copy of the cursor that records in the member are accessed and the current record position is maintained. The inactive cursor in a member is built either directly over a single data space (physical member) or over a data space index, which is over one or more data spaces (keyed physical or logical member).

The ODP (open data path) is the primary control block of the member and is contained in the associated space of the cursor. It contains linkage pointers (to the other objects constituting the member to the FCB, and to other members), and information used by manipulation functions of both the common data management and the data base components.

4 Every physical file member contains a data space, which is a machine interface object that holds records in arrival order sequence. (When describing the fields of the data space entry to the machine interface, the data base component uses the format description linked to the physical file.) A pointer back to the member's primary object (the cursor) is maintained in the associated space of the data space.

5 Every member of a keyed file (physical or logical) contains a data space index, created according to the file's access path specifications (keys, alternate collating sequence, select/omit specifications, and so forth). For a physical member, the index creates an apparent keyed sequence for the records in the member's data space; for a logical member, the index appears to merge and sequence the records from all the based-on physical members in its scope list. In the associated space of the data space index there is a pointer to the cursor.

6 A file's members form a doubly linked chain from the FCB, in the order of their addition to the file. The FCB has first- and last-member pointers; each member has a previous- and a next-member pointer.

7 Directory objects contain indications of the sharing (dependencies) between data base files. A file, a format or a member can have a directory that contains a list of pointers to files or members that depend on it. There are five kinds of directories:

A A format directory belongs to a format and addresses all files that use that record format. If a format is issued by only one file, a directory is not created for that format.

B A file shared-data directory belongs to a physical file and addresses all logical and derived files that are based on that physical file.

8 Join logical files are joined together using data space indexes. There is one DSI for each secondary piece of a joined logical file. Join logical files will use existing indexes by implicitly sharing existing indexes.

Note: If a joined logical file is keyed, it will have an additional access path ordering the primary.

C A member shared-data directory belongs to a physical member and addresses all logical and derived members that are based on that physical member.

D A file shared-access-path directory belongs to a keyed file and addresses all derived files that are defined to share the access path of that keyed file.

E A member shared-access-path directory belongs to a keyed member and addresses all derived members that use the index of that member.

F An implicit-shared-access path directory belongs to an access path and addresses all of the members which implicitly share an index.

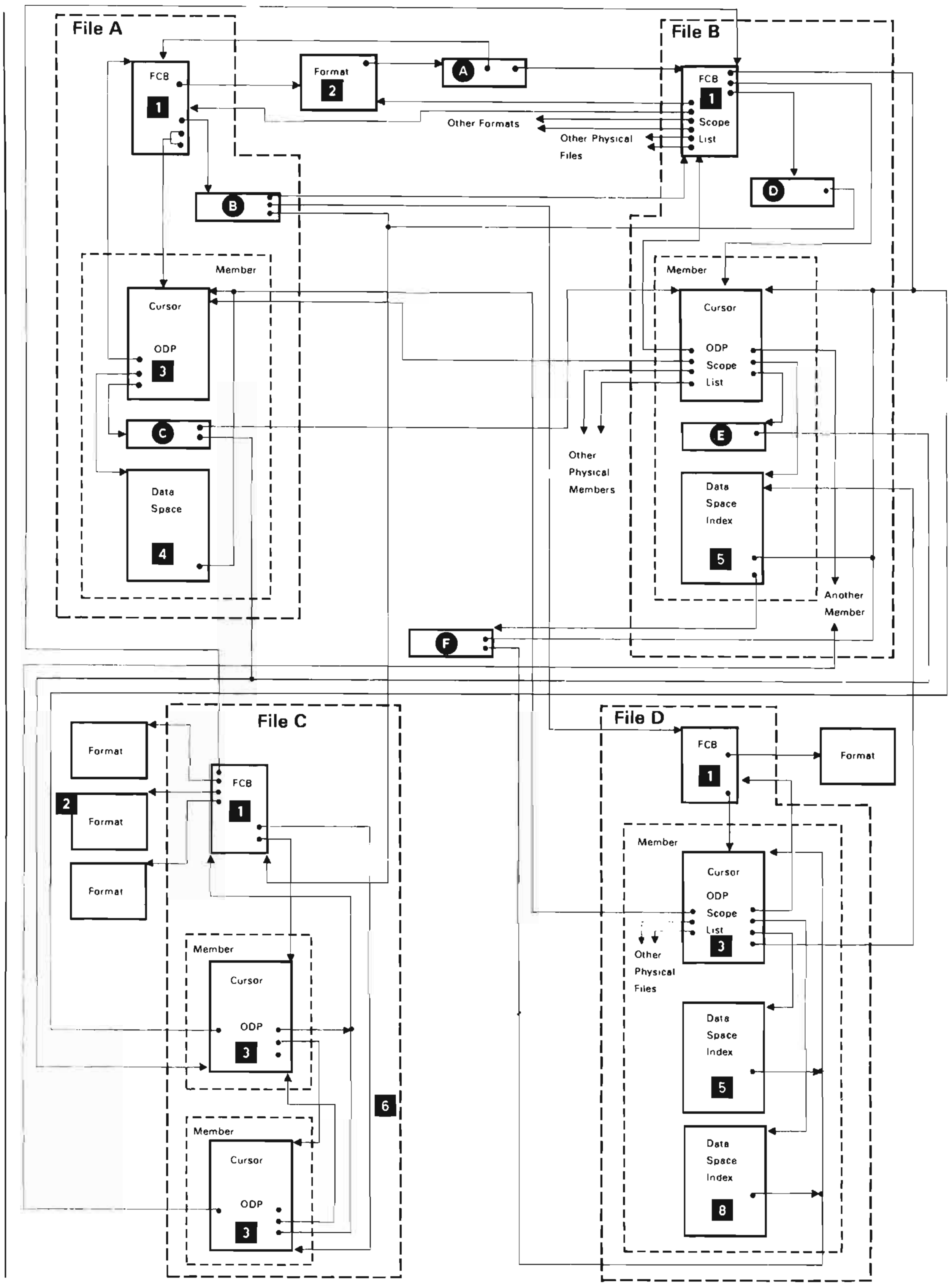


Figure DB-2. The Structure of Data Base Files

This page is intentionally left blank.

Structure of an Open Data Base Member

Figure DB-3 shows the general structure of a data base ODP after a data base file member has been opened. The QDMCOPEN module of common data management creates the ODP by issuing a machine interface Create Duplicate Object instruction while referencing a permanent data base member. The entire data base ODP is implemented in the associated space of the cursor. To aid in performance, the ODP and cursor are copied into the process access group by QDMCOPEN. A field in the root section of the ODP (WWODPROT) tells QDMCOPEN how much of the associated space of a permanent data base member (cursor) to copy. This length can be longer or shorter than the actual permanent copy of the ODP in the member. This length will include enough storage for one I/O buffer; buffer allocation is not performed until open. The figure shows the exact physical location of each section. Because all sections are located by offsets saved in the root section, any section could appear in any place with no effect on the code.

1 WWODPROT: ODP-ROOT Section. This section contains offsets from the start of the associated space to the following sections.

- WWODPOFB
- WWODPIOF
- WWODPLKL
- WWDBODP

It also contains flags and fields common to all devices and data base.

2 WWODPOFB: Open Feedback. This is both a user area (referenced by a pointer set in the UFCB) and a system implementation area. The second half of the open feedback area defines the offsets to the data base I/O module pointers within the system entry point table.

3 WWODPIOF: I/O Feedback Area. This is a user area (referenced by a pointer set in the UFCB) that contains information about each get and put issued to the file.

4 WWODPDBF: Data Base Specific I/O Feedback. This section is located by an offset saved in the common I/O feedback area WWODPIOF.

For a non-keyed data base file, this area is a fixed length. It contains the relative record number and member number of the record just manipulated by the get and put modules.

For a keyed data base file, a key buffer is provided. The buffer is large enough to hold the largest key value for the file. The relative record number and member number are also provided.

5 WWODPLKL: ODP Lock List. This section exactly matches the machine interface template for the machine instruction Lock and Unlock.

Each data space associated with this data base member has a system pointer in the lock list. In addition, one system pointer to the permanent member is provided. The default lock states for all the data spaces and the member are set by QDMCRODP. The default is LSRD if open for input and LSUP if open for update, delete, or output. If an entry for a data space is inactive, the associated format name was moved out to the open data space.

6 WWDBODP: Data Base Section ODP. This section is the key area referred to by all of the data base manipulation modules.

An overlay include (WWDBSTCR) defines the machine interface template for the machine instruction Set Cursor. Enough storage is always provided for a cursor option list (machine interface template) to have 32 data space numbers in the data space search list.

Save areas are provided for the I/O modules to save the current set of inputs (get and put option list and the control list).

An array of record format names is provided with corresponding associated data space numbers. The data space search list provides sets of data space numbers for each record format that spans more than one data space.

7 Record Buffers: One record buffer is automatically allocated by the Create Duplicate Object instruction in QDMCOPEN. If additional buffer space is needed (indicated by opening for input and output or by specifying SEQONLY(*YES)), QDBOPEN extends the associated space to compensate for this.

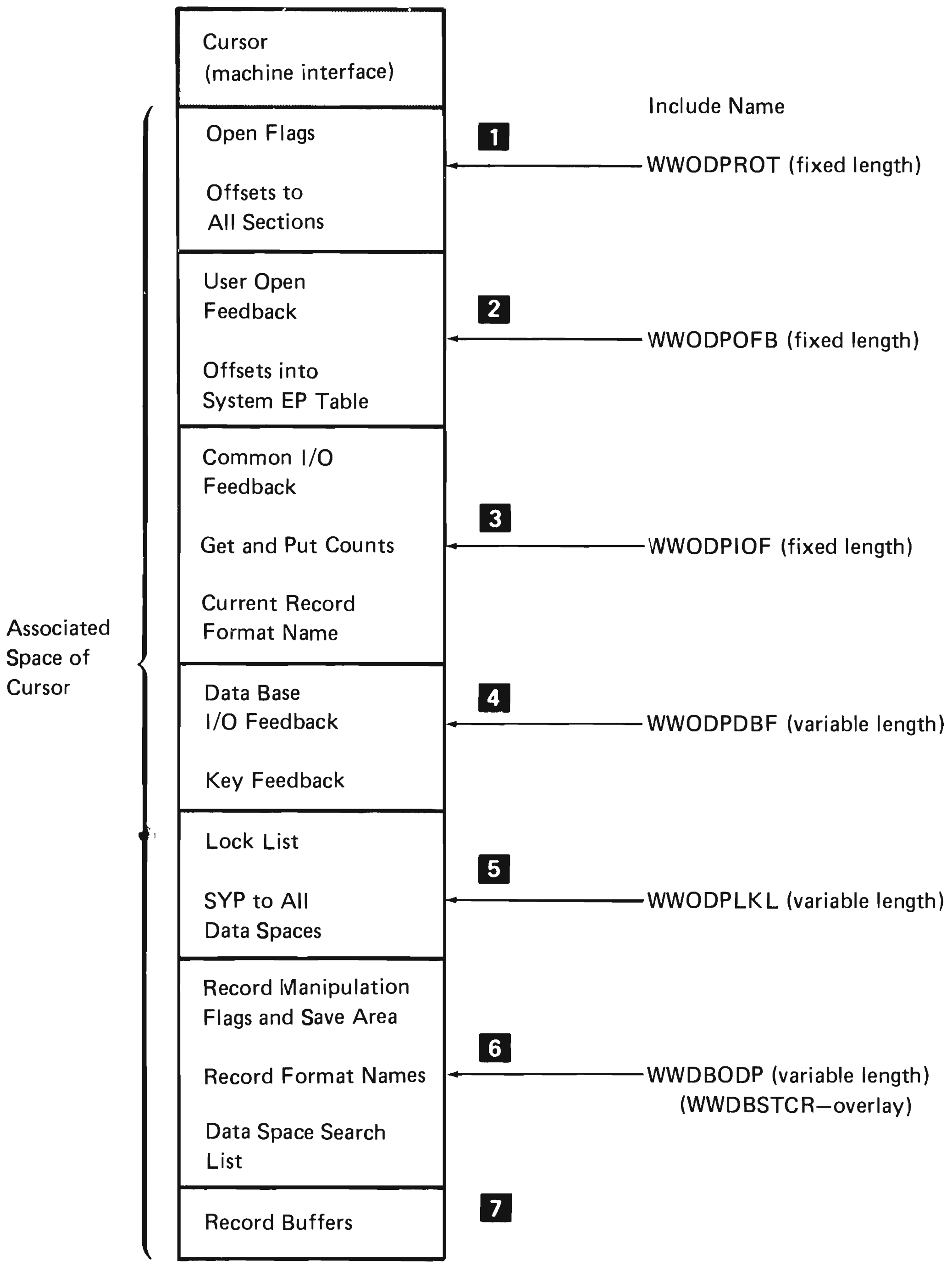


Figure DB-3. Open Data Path (After Open)

Data Base Recovery

Data base recovery, which is performed during IMPL (internal microprogram load), recovers from an abnormal system termination by doing the following:

1. Completes, backs out, or ensures objects' soundness for any data base definition or generic function that was in progress at system failure.
2. Sends messages to the history log about members that were open (partially updated) at system failure and about damaged members.
3. Displays all indexes needing recovery and allows the user to override the recovery option for this IMPL.
4. Rebuilds the data space indexes of keyed members whose access paths were defined as immediate or delayed maintenance, and are in need of recovery.

Information about functions and members requiring recovery is found in objects in the QRECOVERY library. The following includes describe the contents of those objects:

WWWCMISR: This include duplicates the structure of the MISR (machine initialization status record), which contains a list of partially updated and damaged data spaces and data spaces indexes. The MISR (machine initialization status record) is materialized into a space object in QRECOVERY by work control at IMPL, before its call to data base recovery. Also, pseudo-MISR objects are created by QDBFFCPY to list data space indexes that it invalidates and plans to rebuild. Then, if they are interrupted by a system failure, data base recovery finds the pseudo-MISR and rebuilds the indexes.

DBDRCB: This include describes a space object in QRECOVERY that is called a data base definition recovery object. It is created by an invocation of a data base definition or generic function and is left in QRECOVERY if the function is interrupted. In the object, data base recovery finds the recovery program to call for the function. The recovery program receives the data base definition recovery object, uses its identification of the target file to complete or back out the function, and destroys the data base definition recovery object.

WWDBRCOB: This include describes a permanent object in QRECOVERY named QDBMISR. Its logical counter field is used to generate unique names for MISR objects. It also contains addressability to a temporary space used for communication between the two phases of data base recovery.

WWDBRBLD: This describes an independent index created by QDBRCIPS and processed by QDBRCDYN. It contains a list of data space indexes to be rebuilt (those of keyed members with RECOVERY [*AFTSTRCPF]) and MISR objects to destroy (those that data base recovery has finished recovering).

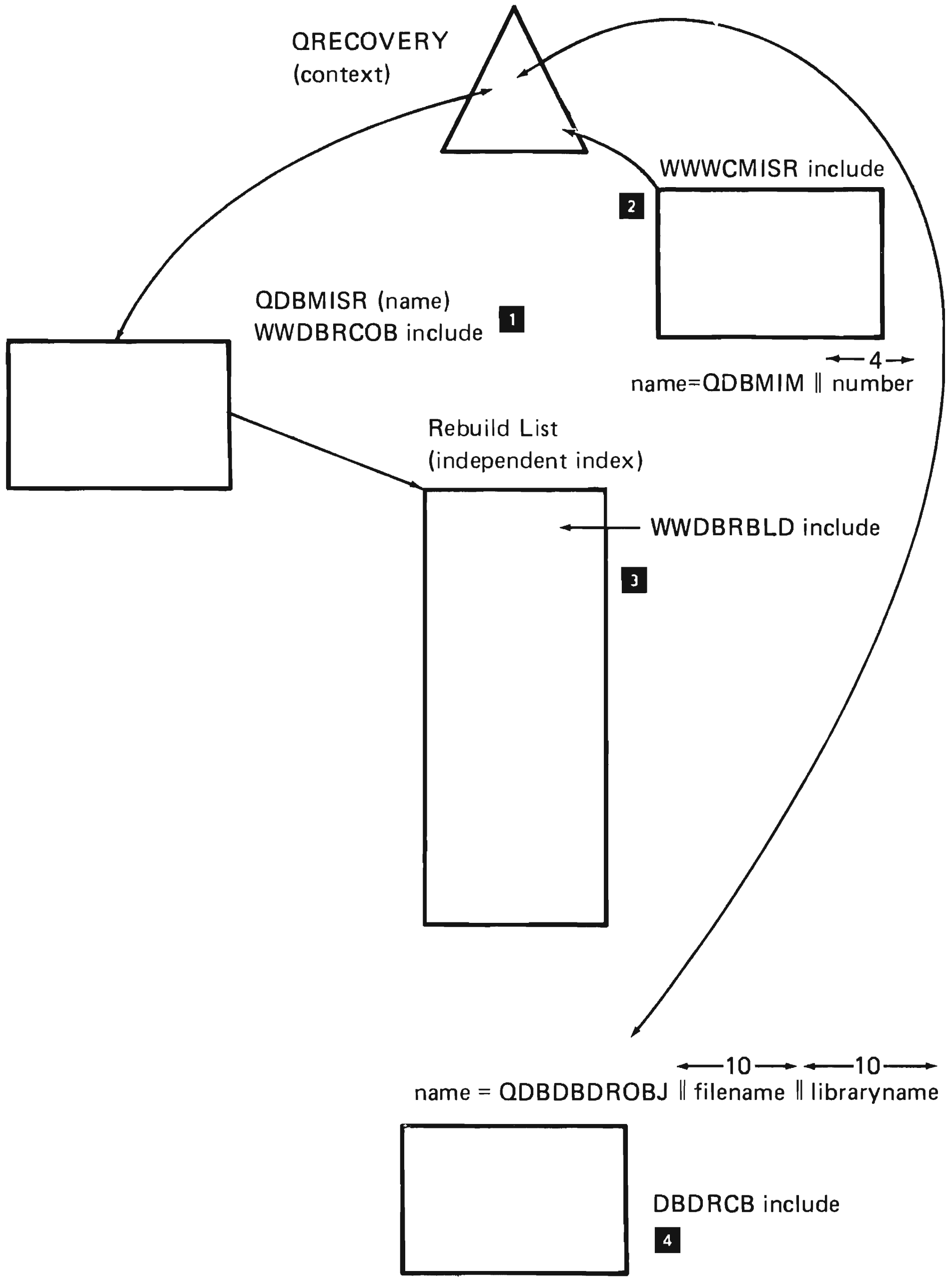


Figure DB-4. Data Base Recovery Objects

Data Base Object Locking

When two operations refer to and update the same control blocks, concurrent invocations of the operations must be controlled to prevent unpredictable results. Data base modules control concurrency by obtaining machine interface locks on the objects that constitute data base files and members and on the various related objects. Locking is a complex topic for the data base component because files and members are composite objects and because of the interrelationships maintained. Not all data base modules need to lock objects; the following covers only the modules that obtain locks.

The definition and extraction functions of the data base component lock only the external user data base objects: files, formats, and members. Figure DB-5 shows the levels of locks obtained by definitional functions on particular objects. When a file is locked, the machine interface lock is actually obtained on the file control block space object; a member lock is a lock on the member's primary object, the machine interface cursor.

Figure DB-5 also shows the generic functions locking machine interface objects that they operate on and that may be shared by members of dependent files.

The manipulation, recovery, and event handling functions of the data base component do their locking on the internal objects that are linked to form a data base file member as well as the FCB. Figure DB-5 and the following text explain the locks they obtain on data spaces, data space indexes, cursors, and FCBs.

Of the open/close and I/O functions, only open obtains object locks. These locks are held until the member is closed and therefore are in force during the I/O operations against the open member. The open locking and close unlocking are performed by modules of the common data management component.

Many of the locks shown in Figure DB-6 are necessary for handling machine interface data space indexes, which exist one-for-one with keyed members. At the machine interface, an index is either valid or invalid; a valid index is one whose keyed access path reflects all the latest record changes to the underlying data spaces. An invalid index can become valid by rebuilding the index. The data base component sees the index as a piece of a member whose maintenance is defined as either immediate, rebuild, or delayed, where immediate or delayed maintenance means the index should never remain invalid.

- QWCISCFR detects the invalid index event when an index has become invalid and invokes QDBIVLIX to rebuild it if it is part of an immediate or delayed maintenance member.
- Data Base Recovery processes those indexes that were marked invalid because they were being updated when the crash occurred. It rebuilds those with immediate or delayed maintenance members and recovery of *STRCPF or *AFTSTRCPF.
- A cursor cannot be activated over an invalid index, so QDBOPEN will rebuild the index of a member with rebuild maintenance or a member whose index is in need of recovery.
- The copy of a physical member's data space is performed by machine interface instructions that cannot succeed when there are valid indexes over the data space. Therefore, QDBFFCPY, if necessary, invalidates indexes and then rebuilds them after copying.

			Member			Parent or Based-Ons				Owner		
	Library	File	Cursor	Index	Data Space	File	Member	Format Share File	Directory Pointer	Old	New	To Library
QDBCHGFI		LENR		LEAR[14]	LEAR[14]				LEAR[15]			
QDBCHGME		LSRD	LEAR						LSRO[13]			
QDBCHKFI		LSRO										
QDBCLRPF		LSRD	LSRD		LENR							
QDBCRTFI	LSUP	LENR				LSRD		LSRD				
QDBCRTME		LEAR		LEAR[17]		LSRD	LSRD					
QDBDIRUP									LEAR			
QDBDLTFI	LSUP	LENR		LEAR[19]					LEAR			
QDBDLTME	LSUP	LEAR	LENR	LEAR[18]					LEAR			
QDBDMPFI		LSRO										
QDBDUPFI[14]		LENR				LSRD						
QDBEXDFI		LSRO										
QDBEXDME		LSRO							LSRO			
QDBEXTWU		LSRO							LSRO			
QDBFFCPY		[2]	[2]		[3]							
QDBGRTFI		LENR			LSUP[1,7]							
QDBINZPF		LSRD	LSRD		LEAR							
QDBISHRX				LEAR					LEAR[16]			
QDBIVLIX		LSRD		LENR	LSRD							
QDBMOVFI	LSUP	LENR										LSUP
QDBOPEN				LEAR								
QDMCOPEN		LSRD	LSRD		[4,6]							
QDBQUERY		LSRD	LSRD	LEAR	[4,6]							

Figure DB-5 (Part 1 of 2). Data Base Functions Locking

			Member			Parent or Based-Ons				Owner		
	Library	File	Cursor	Index	Data Space	File	Member	Format Share File	Directory Pointer	Old	New	To Library
QDBRCIPS through QDBRCDYN[5]		LSRD	LSRD		LSRD[4]							
QDBRCDLK		[2]	[2]		[3]							
QDBRNMFI	LSUP	LENR		LENR[8]	LENR[1,7]							
QDBRNMME		LEAR	LENR	LENR	LENR[1]							
QDBRSPRE		LENR[11]										
QDBRVKFI		LENR			LENR[1,7]							
QDBSIZFI		LSRO										
QDBSVPRE		LSRO[13]			LSRO[9,11]							
QDBXFRFI		LENR		LSUP[8]	LSUP[1,7]					LSUP	LSUP	

[1] This function locks the member's sole data space for a physical member (for each member of the file if the function is file oriented). If it operates on a logical file, it locks no data spaces.
[2] Before this function is invoked, the member must be open (see QDMCOPEN locks); then locking for data base copy may occur.
[3] This function locks the to-member's-data-space (LENR) and the from-member's-data-space (LEAR).
[4] This function locks data spaces under the member (physical or logical).
[5] The locks necessary for data base recovery are obtained by QDBRCIPS during IMPL and are held by the start CPF process until QDBRCDYN completes the recovery function (after CPF is up and running).
[6] Data space lock states obtained by OPEN for a given data space under the member:
- If the UFCB or override explicitly specifies a lock state for the format associated with the data space (RCDFMTLCK), that lock state is obtained;
- If open for update, delete, and/or output, then the data space is locked (LSUP) as a default;
- Otherwise the data space is locked (LSRD) as a default.

[7] The security functions lock the data spaces of a physical file's members only if there are logical files over the physical files.
[8] QDBXFRFI also locks the data space indexes of a keyed file's members only if there are logical derived files over the keyed file.
[9] Dependent files are also locked (LENR).
[10] The file and data space are locked (LENR) if saved with storage freed and there is logical dependency on the file.
[11] Save/restore locks the library, file, and potentially the data spaces or dependent logical files. If the system is in a restricted state during the save/restore, nothing is locked.
[12] This function locks the data space index only if the file is nonderived and keyed.
[13] Directory is only locked if EXPDATE or FRCRATIO is changing for nonderived and keyed members.
[14] The data spaces are locked (LEAR) only if the data space attributes are changing (SIZE,ALLOCATE,UNIT, and DLTPCT parameters). The data space indexes are locked (LEAR) only if the data space index attributes are changing (MAINT, FRCACCPTH, RECOVER, and UNIT parameters).
[15] Directory is locked (LEAR) if EXPDATE or FRCRATIO is changing for nonderived and keyed members. Implicit access sharing directory is locked (LEAR) if any of the indexes are implicitly shared.
[16] When called by QDBCRTME to find a sharable index, if a sharable index is found, the DSI is locked (LEAR) and the SPP to the implicit access sharing directory is locked (LEAR).
[17] All newly created logical indexes are locked (LEAR).
[18] If the DSI is implicitly shared, it is locked (LEAR).
[19] All logical indexes are locked (LEAR).

Figure DB-5 (Part 2 of 2). Data Base Functions Locking

INTRODUCTION

The device configuration component of the CPF (control program facility) has the following functions:

- Describes the features and characteristics of control units, communications lines, and devices to the system. These descriptions are supplied by user-defined parameters that have passed from the command analyzer to a command processing program in device configuration. The command processing programs can create, change, delete, and display the descriptions.
- Describes the installation-dependent tables, such as translate tables, collating sequence tables, and print belt image tables to the system. Tables can be created (from user-defined source files) or deleted by device configuration command processing programs.
- Describes the five user-defined edit codes to the system. Device configuration command processing programs can create, delete, and display the edit code descriptions.
- Controls the power status of some devices and control units and the online or offline status of devices, control units, and lines through the use of other device configuration command processing programs and modules.

GENERAL OVERVIEW

Device Configuration Modules

To provide the previously mentioned functions, the following device configuration modules are used. The modules are grouped by the functions that they provide.

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QDCCRLUD–Create Device Description (CRTDEVD)[1]: This module checks the input parameters and routes the input to the proper module to build the LUD (logical unit description) template.

  QDCCCARD–Create Device Description for Card Devices: This module builds the template and creates the device description for a card device.

  QDCCDSKT–Create Device Description for Diskette Device: This module builds the template and creates the device description for diskette devices.

  QDCCINST–Create Device Description for Install From Save/Restore: This module creates the device description from save/restore through the install function.

  QDCCPRNT–Create Device Description for Printer Devices: This module builds the template and creates the device description for a system printer.

  QDCCSDLC–Create Device Description for a Remote SDLC Device: This module builds the template and creates the device description for remote SDLC work station display and printer devices.

  QDCCSLU1–Create Device Description for LU-1 Secondary: This module builds the template and creates the device description for a secondary LU-1 device.

  QDCCTAPE–Create Device Description for Tape Devices: This module builds the template and creates the device description for tape devices.

  QDCCWSC–Create Device Description for WSC Devices: This module builds the template and creates the device description for work station display and printer devices attached to a local WSC (work station controller) or WSC-E (work station controller extended).

[1]This module is a CPP (command processing program).

QDCCRBSC–Create Device Description for BSC Device: This module builds the template and creates the device description for BSC and BSCT (BSC multipoint tributary) devices.

QDCCPEER–Create Device Description for Peer Device: This module builds a template and creates a device description for peer devices (for advanced program-to-program communications). An event is signaled to the logical unit services process for each successfully created device description.

-->QDCCRCD–Create Control Unit Description (CRTCUD)[1]: This module checks the input parameter and routes the input to the proper module to build the CD template.

QDCCDWSC–Create Control Unit Description for WSC: This module builds the template for WSC (or WSCE) and signals the create CD event to invoke QDCCTCND to create the CD.

QDCCDWS–Create Control Unit Description for Remote Work Station: This module builds the template for remote work station control units, and signals the create CD event to invoke QDCCTCND to create the CD.

QDCCDTAP–Create Control Unit Description for Tape Controller: This module builds the template for tape controllers and signals the create CD event to invoke QDCCTCND to create the CD.

QDCCDSLU–Create Control Unit Description for PU2 Secondary: This module builds the template for PU2 secondary controllers and signals the create CD event to invoke QDCCTCND to create the CD.

QDCCDINS–Create Control Unit Description for Install from Save/Restore: This module creates the control unit description from save/restore through the install function, and signals the create CD event to invoke QDCCTCND to create the CD.

QDCCDBSC–Create Control Unit Description for BSC Controller: This module builds the template for BSC and BSCT controllers and signals the create CD event to invoke QDCCTCND to create the CD.

QDCCDPER–Create Control Unit Description for Peer Controller: This module builds a template for peer controllers and signals the create CD event to invoke QDCCTCND to create a CD.

QDCCTCND–Create CD Event Handler: This module performs the actual create CD using the template built by the CD type-dependent modules. QDCCTCND runs under the system arbiter process.

-->QDCCRND–Create Line Description (CRTLIND)[1]: This module changes the system configuration to include a new teleprocessing line.

-->QDCADMOD–Add Device Mode Entry (ADDDEVMODE)[1]: This module adds a device mode entry to an existing peer device.

-->QDCCHLUD–Change Device Description (CHGDEVD)[1]: This module changes the attributes of a device.

QDCCGLUD–Change Device Description Event Handler: This module handles any events signaled by the change device description module. It also changes the actual device description. QDCCGLUD runs under the system arbiter process.

-->QDCCHCD–Change Control Unit Description (CHGCUD)[1]: This module changes the attributes of a control unit.

-->QDCCHND–Change Line Description (CHGLIND)[1]: This module changes the attributes of a teleprocessing line.

-->QDCCHMOD–Change Device Mode Entry (CHGDEVMODE)[1]: This module changes the attributes of a peer device's mode entry.

-->QDCDLLUD–Delete Device Description (DLTDEVD)[1]: This module removes a device from the system configuration.

QDCDTLUD–Delete Device Event Handler: This module handles any events signaled by QDCDLLUD. It also does the actual delete. For peer devices, it signals an event to the logical unit services process.

[1]This module is a CPP (command processing program).

-->QDCDLCD–Delete Control Unit Description (DLTCUD)[1]: This module removes a control unit from the system configuration.

QDCDTCD–Delete Control Unit Event Handler: This module handles any events signaled by the QDCDLCD. It also does the actual delete.

-->QDCDLND–Delete Line Description (DLTLIND)[1]: This module removes a teleprocessing line from the system configuration.

-->QDCDSLUD–Display Device Description (DSPDEVD)[1]: This module displays the description of a device to the requestor.

-->QDCDSCD–Display Control Unit Description (DSPCUD)[1]: This module displays the description of a control unit to the requestor.

-->QDCDSCST–Display Channel Status (DSPCHLSTS): This module displays the logical channel status for X.25 lines.

-->QDCDSLST–Display Link Status (DSPLNKSTS): This module displays the logical link status for X.25 lines.

-->QDCDSND–Display Line Description (DSPLIND)[1]: This module displays the description of a teleprocessing line to the requestor.

-->QDCDSCFG–Display Device Configuration (DSPDEVCFG)[1]: This module displays the configuration of the entire system to the requestor.

-->QDCDSMOD–Display Mode Status (DSPMODSTS)[1]: This module displays status information related to a peer device's mode entry.

-->QDCDSSTS–Display Network Status (DSPLINSTS, DSPCTLSTS, and DSPDEVSTS)[1]: This module displays the status of selected device configurations on a system, displays the jobs using active devices, and allows input to request additional displays, vary capability, and job cancellation.

QDCHNCMD–Handle Command Input: This module handles the functions requested from input to the screen displayed by QDCDSSTS.

QDCVANET–Vary Network: This module handles line name or control unit name input, and calls QDCVALUD, QDCVARCD, and QDCVARND to vary an entire network online or offline.

QDCCHPMT–Change Prompt: This module provides a prompt for the Change Line Description (CHGLIND) command, Change Control Unit Description (CHGCUD) command, or Change Device Description (CHGDEVD) command with existing and valid values.

-->QDCCRTBL–Create Table (CRTTBL)[1]: This module defines a 256-byte table to the system.

-->QDCCRPRI–Create Print Image (CRTPRTIMG)[1]: This module defines the print image for a print belt to the system. If the BELTNBR parameter is used, the appropriate translate table is also built by QPNCPITT.

-->QDCXLATE–High-Level Language Interface to Translate Tables: This module issues the machine instructions to perform byte-by-byte translation of fields passed to it by a high-level language.

-->QDCCRECD–Create Edit Description (CRTEDTD)[1]: This module creates the description of a user-defined edit code as specified by the command parameters.

-->QDCDSECD–Display Edit Description (DSPEDTD)[1]: This module displays the edit code description to the requestor.

-->QDCEDITS–Edit Code Expansion for Standard Edit Codes: This module creates an edit mask for standard edit codes. The mask is used by the machine edit instructions.

QDCEDITU–User Edit Code Expansion: This module creates an edit mask for user-defined edit codes. The mask is used by the machine edit instructions.

-->QDCEDITW–Edit Word Expansion: This module creates an edit mask to be used by the machine edit instructions.

-->QDCINIT–Device Initialization Interface: This module builds a parameter list and calls the appropriate CPPs for the devices, teleprocessing lines, and control units attached to the system.

[1]This module is a CPP (command processing program).

-->QDCSHUTD–Device Shutdown Interface: This module builds lists of the active online or powered-on device descriptions, control units, and teleprocessing line descriptions for use by QDCVALUD, QDCPWLUD, QDCVARCD, QDCPWCUD, and QDCVARND.

-->QDCLUDCF–LUD Device Failure Event Monitor: This module runs under the system arbiter process. If the LUD device failure event occurs, this module performs error recovery and sends the appropriate messages.

-->QDCLUDRC–LUD Device Failure Message Reply Handler: This module runs in the system arbiter process. QDCLUDRC processes replies to any inquiry message sent by QDCLUDCF and performs the appropriate recovery actions.

-->QDCPWLUD–Power Device: This module passes a list of device names that are to have their power status changed to QDCPRLUD.

QDCPRLUD–Power LUD: This module is an event handler that executes under control of the system arbiter to change the power status of devices specified in the Power Device command.

-->QDCPWCUD–Power CUD: This module changes the power status of control units specified in the Power Control Unit command.

-->QDCLRFMT–LUD ASP Reformat Modules: This module is used to reformat the LUD associated space (ASP) to extend the device dependent spaces without requiring the user to delete and re-create the existing device descriptions. This module is called by QDCINIT at start CPF time.

-->QDCRSCDR–Reset IGC Controller RAM: This module is used by macro RSCDRAM. It is an interface provided for character generation utility (CGU). It resets WSC-E with IGC RAMs and the 5294 control unit.

-->QDCVALUD–Vary Device (VRYDEV)[1]: This module places the specified device in an online or offline state. It also does the actual vary off.

QDCVRLUD–Vary Device Event Handler: This module handles the events signaled by QDCVALUD. It also does the actual vary on.

-->QDCVARCD–Vary Control Unit (VRYCTLU)[1]: This module places the specified control unit an online or offline state.

QDCVARND–Vary Line (VRYLIN)[1]: This module places the specified teleprocessing line in an online or offline state.

Note: When creating device configuration descriptions, the following sequence should be followed. The descriptions can be created out of sequence, but any references to names of descriptions not yet created will be rejected.

1. Create line descriptions.

2. Create control unit descriptions.

3. Create device descriptions.

[1]This module is a CPP (command processing program).

This page is intentionally left blank.

Create Logical Unit Description, Control Unit Description, and Network Description

Figure DC-1 shows an overview of the device configuration create commands operation.

1 The command analyzer decodes a Create Device Description (CRTDEVD) command and control is transferred to QDCCRLUD.

2 QDCCRLUD calls the proper device type-dependent module to build a LUD (logical unit description) template from the user-defined command parameters. It also issues the Create LUD instruction to build the LUD on the system.

- **A** The associated space is initialized.
- **B** For all LUDs, except peer LUDs, the lock is passed to the system arbiter process.
- **C** For peer LUDs only, an event is signaled to the logical unit services process.

3 The command analyzer decodes a Create Control Unit Description (CRTCUD) command and control is transferred to QDCCRCD.

4 QDCCRCD calls the proper device type-dependent module to build a control unit description template from the user-defined command parameters. QDCCRCD signals the create CD event to invoke QDCCTCND.

- **C** QDCCTCND is invoked to issue the Create CD instruction to build the control unit description on the system. QDCCTCND is an event handler that runs under control of the system arbiter process.

5 The command analyzer decodes a Create Line Description (CRTLIND) command and control is transferred to QDCCRND.

6 QDCCRND builds a network description template from the user-defined command parameters and then issues the Create ND instruction to build the network description on the system.

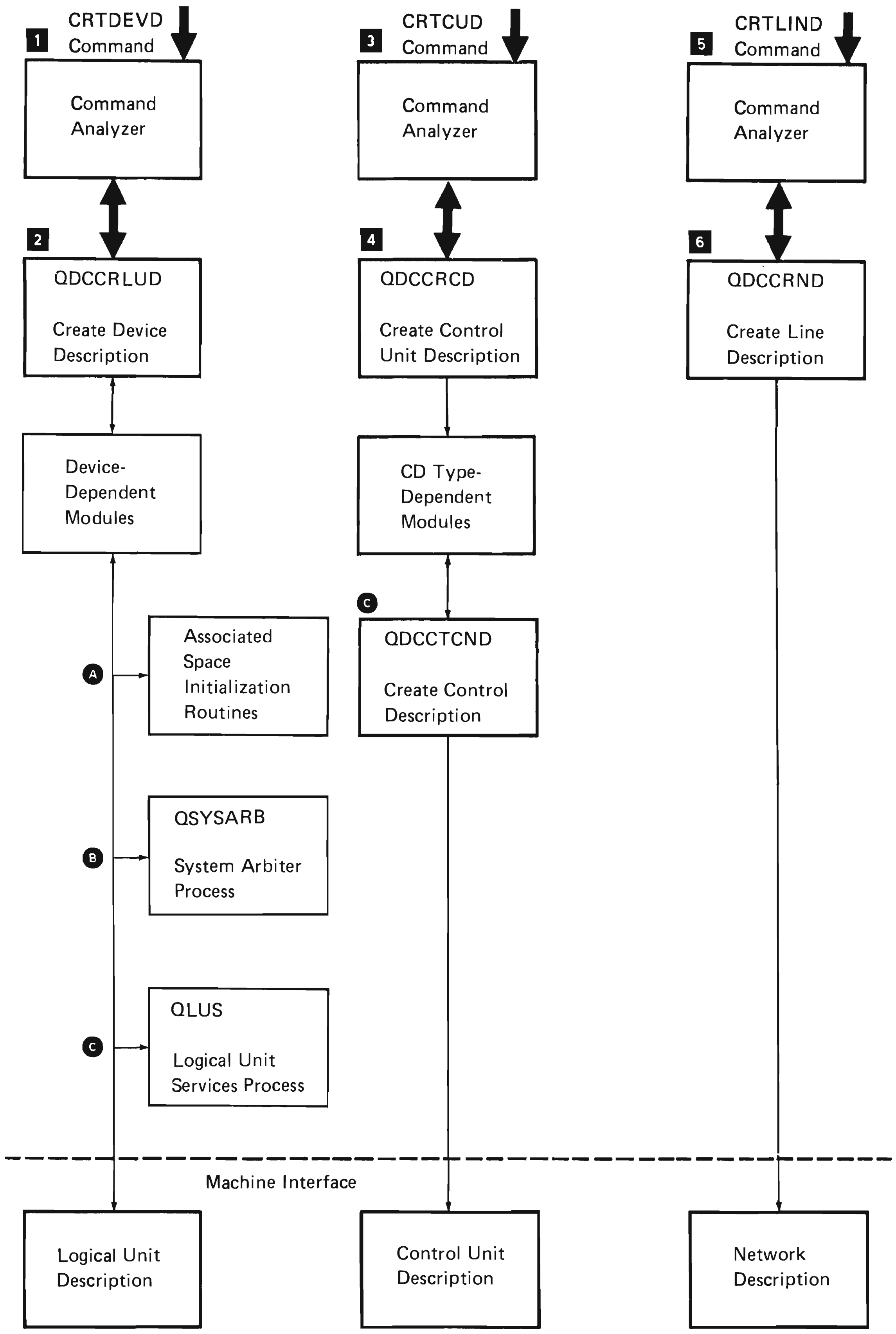


Figure DC-1. Create Logical Description, Control Unit Description, and Network Description Overview

Add Device Mode Entry to a Peer Device Description

Figure DC-2 shows an overview of the device configuration Add Device Mode Entry (ADDDEVMODE) command.

1 The command analyzer decodes an Add Device Mode Entry command and control is transferred to QDCADMOD.

2 QDCADMOD locates the specified peer device description and adds a mode entry to the LUD base on the user-defined command parameters.

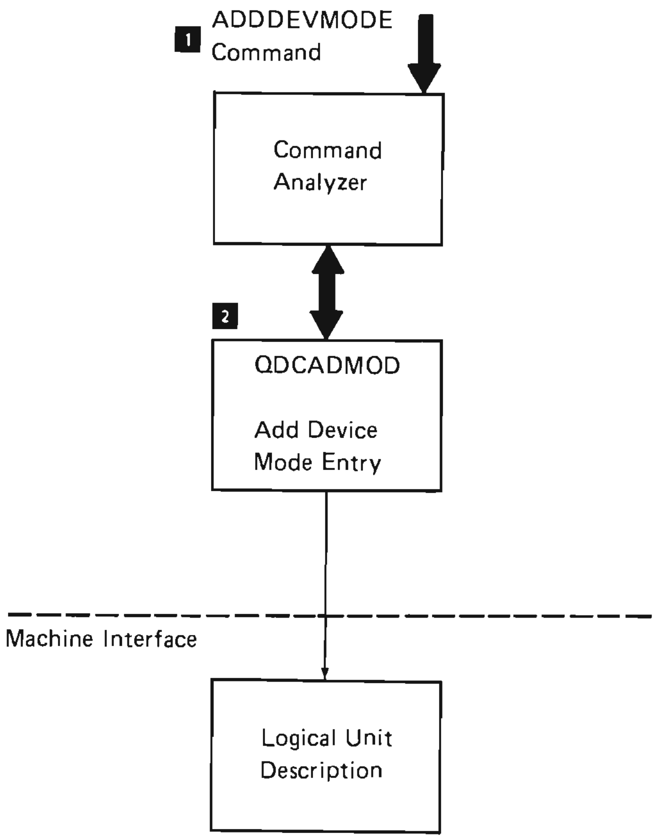


Figure DC-2. Add Device Mode Entry to a Peer Device Description

This page is intentionally left blank.

Delete Logical Unit Description, Control Unit Description, and Network Description

Figure DC-3 shows an overview of the device configuration delete commands operation.

1 The command analyzer decodes a Delete Device Description (DLTDEVD) command and control is transferred to QLIDLOBJ.

2 QLIDLOBJ determines the proper list of parameters to be deleted, based on the user-defined command parameters, and calls QDCDLLUD for each device.

3 QDCDLLUD sets up parameters for QDCDTLUD to delete the associated LUD (logical unit description) from the system.

A If the LUD to be deleted is for a display device, the ?DLTMSGQ macro is issued to QMHDLMSQ to delete the message queue.

B QDCDTLUD is invoked to actually delete the LUD. QDCDTLUD is an event handler that executes under control of the system arbiter process. For peer devices only, an event is signaled to the QLUS process to notify it that the peer device has been deleted.

4 The command analyzer decodes a Delete Control Unit (DLTCUD) command and control is transferred to QLIDLOBJ.

5 QLIDLOBJ determines the proper list of control unit descriptions to be deleted, based on the user-defined command parameters, and calls QDCDLCD for each control unit description.

6 QDCDLCD deletes the associated CD control unit description from the system.

C QDCDTCD is invoked to actually delete the CD. QDCDLCD is an event handler that runs under control of the system arbiter process.

7 The command analyzer decodes a Delete Line Description (DLTLIND) command and control is transferred to QLIDLOBJ.

8 QLIDLOBJ determines the proper list of line descriptions to be deleted, based on the user-defined command parameters, and calls QDCDLND for each line description.

9 QDCDLND deletes the specified network description from the system.

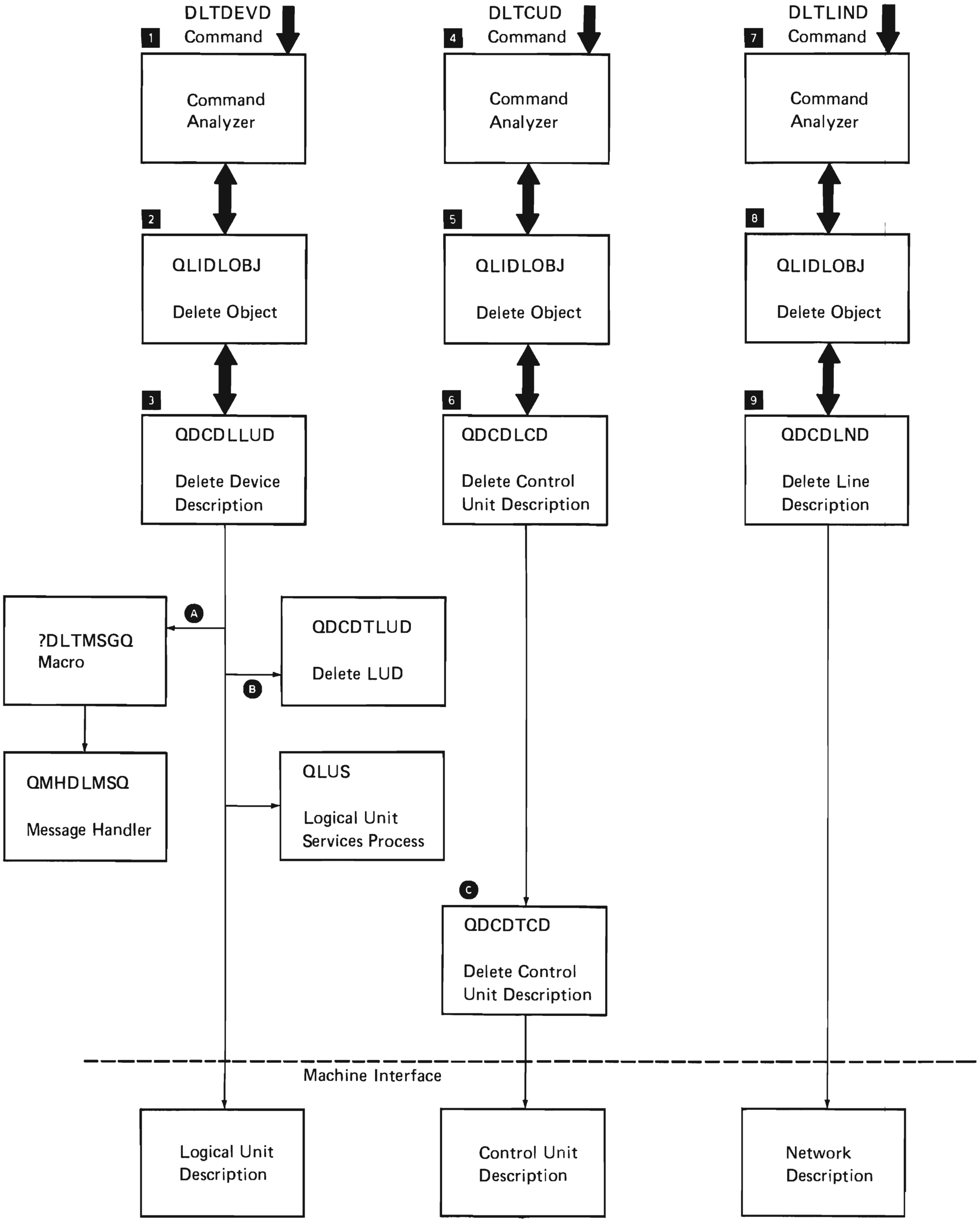


Figure DC-3. Delete Logical Unit Description, Control Unit Description, and Network Description Overview

Change Logical Unit Description, Control Unit Description, Network Description, and Device Mode Entry

Figure DC-4 shows an overview of the device configuration change commands operation.

1 The command analyzer decodes a Change Device Description (CHGDEVD) command and control is transferred to QDCCHLUD.

2 QDCCHLUD invokes QDCCGLUD to modify the specified LUD (logical unit description) using information in the user-defined command parameters that are passed to it by QDCCHLUD. QDCCGLUD is an event handler that executes under control of the system arbiter process.

3 The command analyzer decodes a Change Control Unit (CHGCUD) command and control is transferred to QDCCHCD.

4 QDCCHCD modifies the specified control unit description using information in the user-defined command parameters.

5 The command analyzer decodes a Change Line Description (CHGLIND) command and control is transferred to QDCCHND.

6 QDCCHND modifies the specified line description using information in the user-defined command parameters.

7 The command analyzer decodes a Change Device Mode Entry (CHGDEVMODE) command and control is transferred to QDCCHMOD.

8 QDCCHMOD modifies the peer device's mode entry using information in the user-defined command parameters.

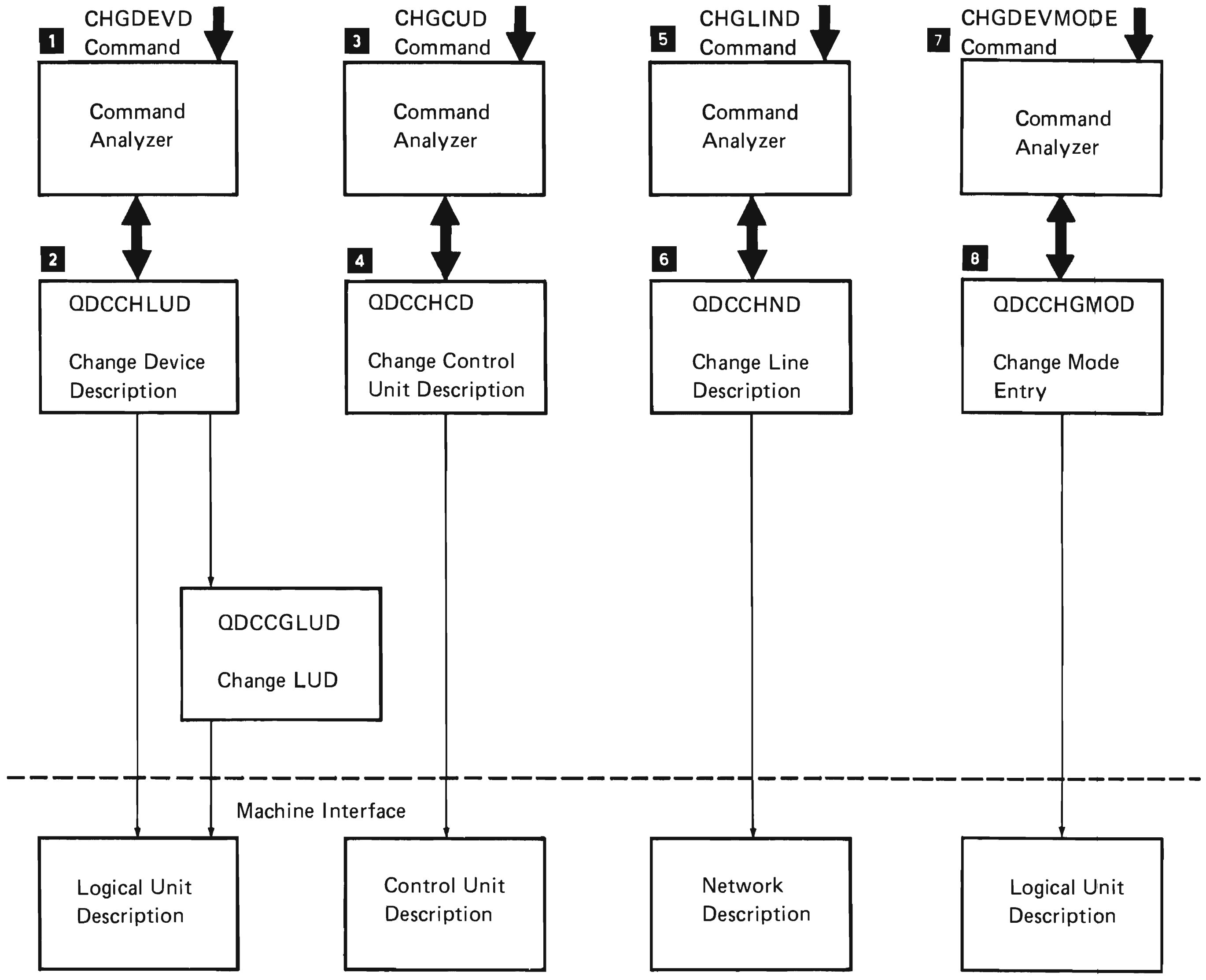


Figure DC-4. Change Logical Unit Description, Control Unit Description, and Network Description Overview

Create, Delete, and Display Edit Codes and Edit Macro Interface

Figure DC-5 shows an overview of the device configuration edit code commands as well as the edit code macro interface operation.

Create, Delete, and Display Edit Code Commands

1 The command analyzer decodes a Create Edit Code Description (CRTEDTD) command and control is transferred to QDCCRECD.

2 QDCCRECD, using information from the command parameters, creates a user-defined edit code.

3 The command analyzer decodes a Delete Edit Code Description (DLTEDTD) command and control is transferred to QLIDLOBJ.

4 QLIDLOBJ deletes the edit code description from the system.

5 The command analyzer decodes a Display Edit Code Description (DSPEDTD) command and control is transferred to QDCDSECD.

6 QDCDSECD displays information about the specified edit code to the user.

Edit Code Macro Interface

7 The ?CRTEDTMS macro provides the macro interface to build the edit masks required for the edit function.

8 QDCEDITS builds edit masks from edit codes passed through the ?CRTEDTMS macro for edit codes other than 5 through 9.

9 If edit codes 5 through 9 are passed, QDCEDITU is called to build an edit mask from the user-defined edit code in the QSYS library.

10 QDCEDITW builds edit masks from edit words passed to it by the ?CRTEDTMS macro.

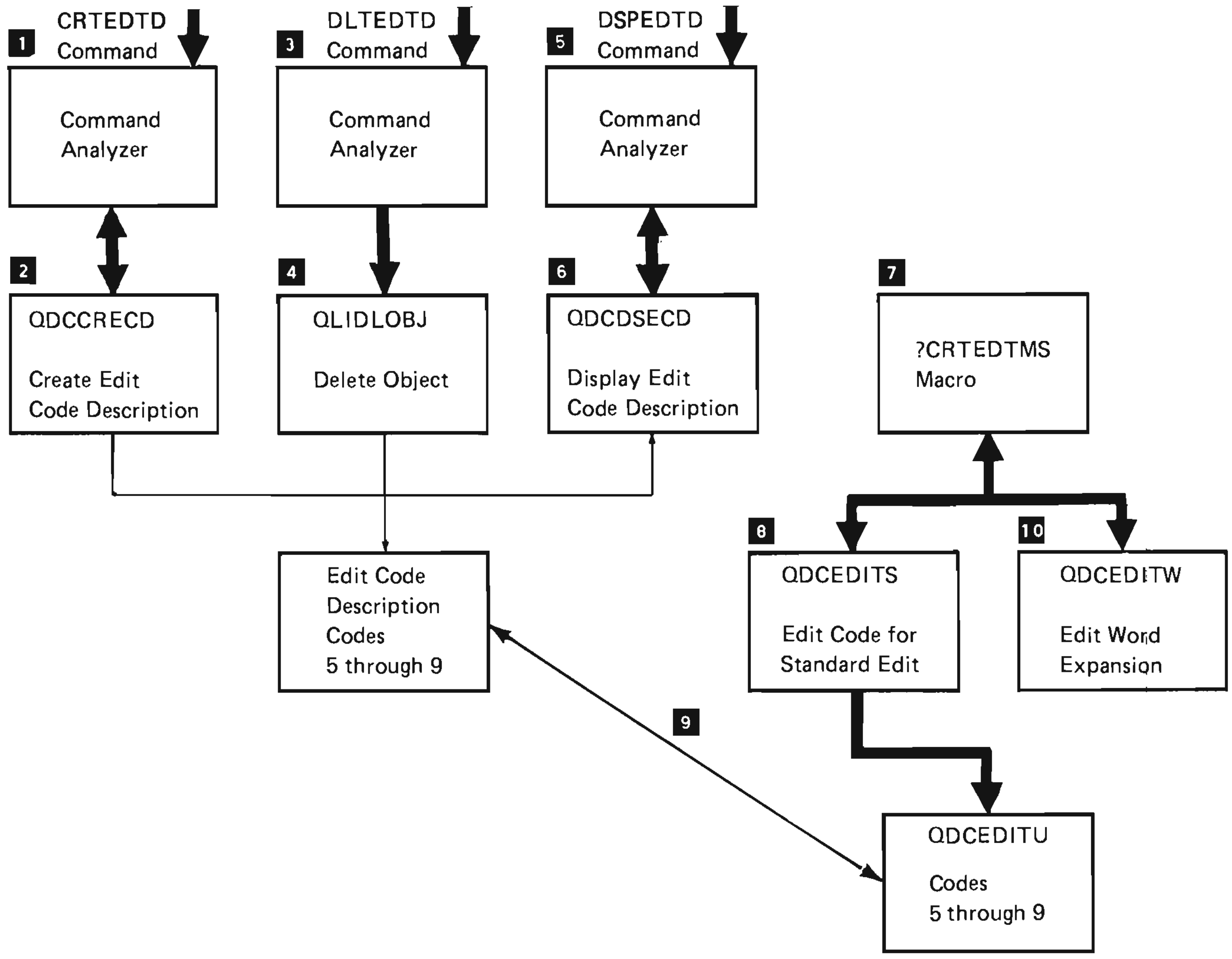


Figure DC-5. Edit Code Commands and Macro Interface Overview

Create and Delete Print Images and Tables

Figure DC-6 shows an overview of the device configuration print image and table commands operation.

Print Image

1 The command analyzer decodes a Create Print Image (CRTPRTIMG) command and control is transferred to QDCCRPRI.

A QDCCRPRI creates a print belt image from user-defined source files, in either hexadecimal or character format, as defined by the header record. The print image can be of various sizes and, unless specified differently, is stored in the QGPL library.

Note: IBM supplies default print images for common print belts. If the BELTNBR parameter is specified, control is transferred to QPNCPITT, which builds the appropriate print image and translate table from the default values for that particular belt number.

1 The command analyzer decodes a Delete Print Image (DLTPRTIMG) command and control is transferred to QLIDLOBJ.

B QLIDLOBJ deletes the print image from the specified library.

Tables

1 The commmand analyzer decodes a Create Table (CRTTBL) command and control is transferred to QDCCRTBL.

C QDCCRTBL creates 256-byte tables as specified by user-defined command parameters. The tables are in hexadecimal format and can be used as translate tables, alternate collating sequence tables, and so forth.

1 The command analyzer decodes a Delete Table (DLTTBL) command and control is transferred to QLIDLOBJ.

D QLIDLOBJ deletes the table from the system.

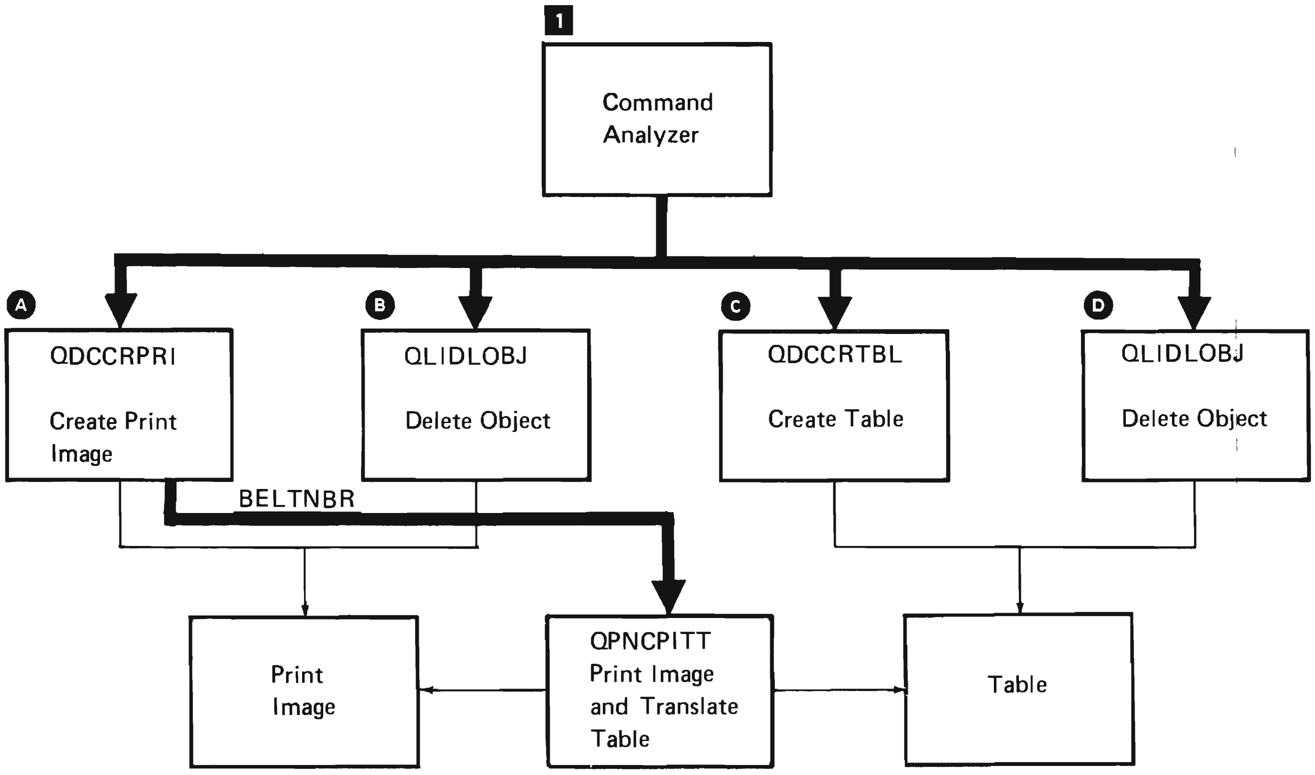


Figure DC-6. Create/Delete Print Image and Tables Overview

Device Configuration Display Commands

Figure DC-7 shows an overview of the device configuration display commands operation.

1 The command analyzer decodes a Display Device Description (DSPDEVD) command and control is transferred to QDCDSLUD. QDCDSLUD then obtains information from the specified LUD (logical unit description) and displays or prints it to the user.

2 The command analyzer decodes a Display Control Unit Description (DSPCUD) command and control is transferred to QDCDSCD. QDCDSCD then obtains information from the specified CD (control unit description) and displays or prints it to the user.

3 The command analyzer decodes a Display Line Description (DSPLIND) command and control is transferred to QDCDSND. QDCDSND then obtains network description information from the specified ND (network description) and displays or prints it to the user.

4 The command analyzer decodes a Display Device Configuration (DSPDEVCFG) command and control is transferred to QDCDSCFG. QDCDSCFG then displays or prints information about all of the devices on the system. There is one record for each ND, showing name, address, and if the description is *not* for a switched network, the attached control units. There is one record for each CD showing name, address, type, attached devices, and if applicable, the attached lines. There is also one record (or more) that shows LUD information: name, address, type, model number, and if applicable, the attached control unit.

5 The command analyzer decodes a Display Mode Status (DSPMODSTS) command and control is transferred to QDCDSMOD. QDCDSMOD then obtains information from the specified LUD and displays or prints it to the user.

6 The command analyzer decodes a Display Channel Status (DSPCHLSTS) command and control is transferred to QDCDSCST. QDCDSCST then obtains the channel status information from the specified ND (network description) and displays or prints it to the user. Channel status can only be displayed for X.25 lines.

7 The command analyzer decodes a Display Link Status (DSPLNKSTS) command and control is transferred to QDCDSLST. QDCDSLST then obtains the link status information from the specified ND (network description) and associated CDs (controller descriptions). The module displays or prints the link status information to the user. Link status can only be displayed for X.25 lines.

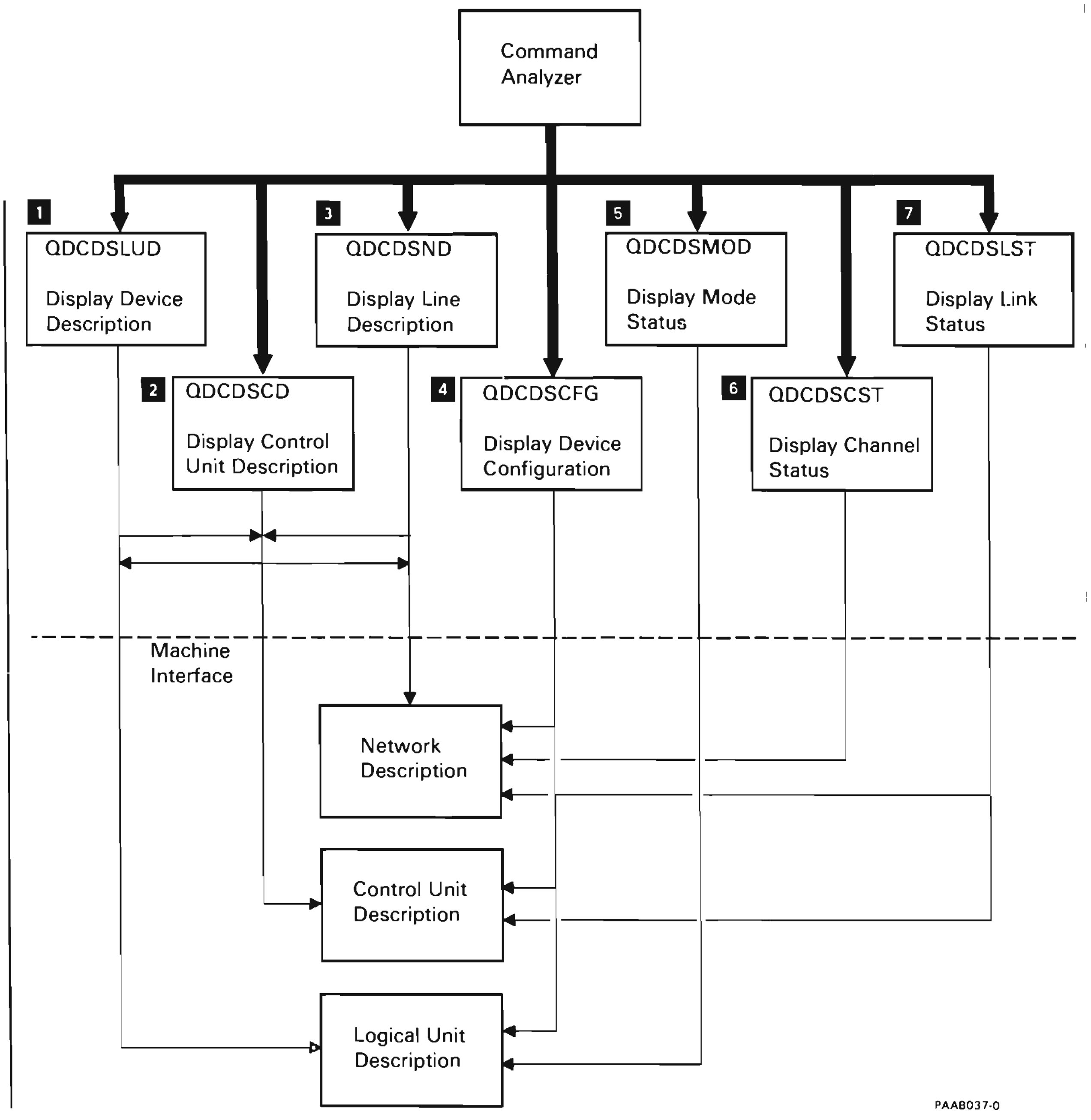


Figure DC-7. Device Configuration Display Commands Overview

Communication Status Display Commands

Figure DC-8 shows an overview of the communication status display commands.

1 The command analyzer decodes a Display Line Status (DSPLINSTS) command, Display Control Status (DSPCTLSTS) command, or Display Device Status (DSPDEVSTS) command, and control is transferred to QDCDSSTS. QDCDSSTS then displays information about the following:

- All lines and their attached control units and devices
- A specific line and its attached control units and devices
- All control units and their attached devices
- A specific control unit and its attached line, if applicable, and devices
- All devices
- A specific device and its attached line and control unit, if applicable

There is one record for each ND and CD, showing name and status. There are one or more records for each LUD showing name, status, and using job name if the status is active.

2 If the output is displayed, then QDCDSSTS calls QDCHNCMD to handle any input. QDCHNCMD calls the following functions based on the command input:

A Display the job using the Display Job, Display Reader, and Display Writer (DSPJOB, DSPRDR, DSPWTR) commands.

B Display the device configuration object in detail (QDCDSND, QDCDSCD, QDCDSLUD)

C Prompt the Change Device Description, Change Control Unit Description, and Change Line Description (CHGDEVD, CHGCUD, CHGLIND) commands for the device configuration object (QDCCHPMT).

D Vary the network off or on starting with an ND or CD (QDCVANET) or vary a single LUD off or on (QDCVALUD)

E Cancel the job using the Cancel Job, Cancel Reader, and Cancel Writer (CNLJOB, CNLRDR, CNLWTR) commands.

Any messages generated by the above are built into a message subfile, and this subfile is returned to QDCDSSTS.

F Display the mode status for a peer device (QDCDSMOD).

G Stop or resume communications recovery for the Stop Line Recovery, Resume Line Recovery, Stop Control Unit Recovery, Resume Control Unit Recovery, Stop Device Recovery, Resume Device Recovery (STPLINRCY, RSMLINRCY, STPCTLRCY, RSMCTLRCY, STPDEVRCY, RSMDEVRCY) commands.

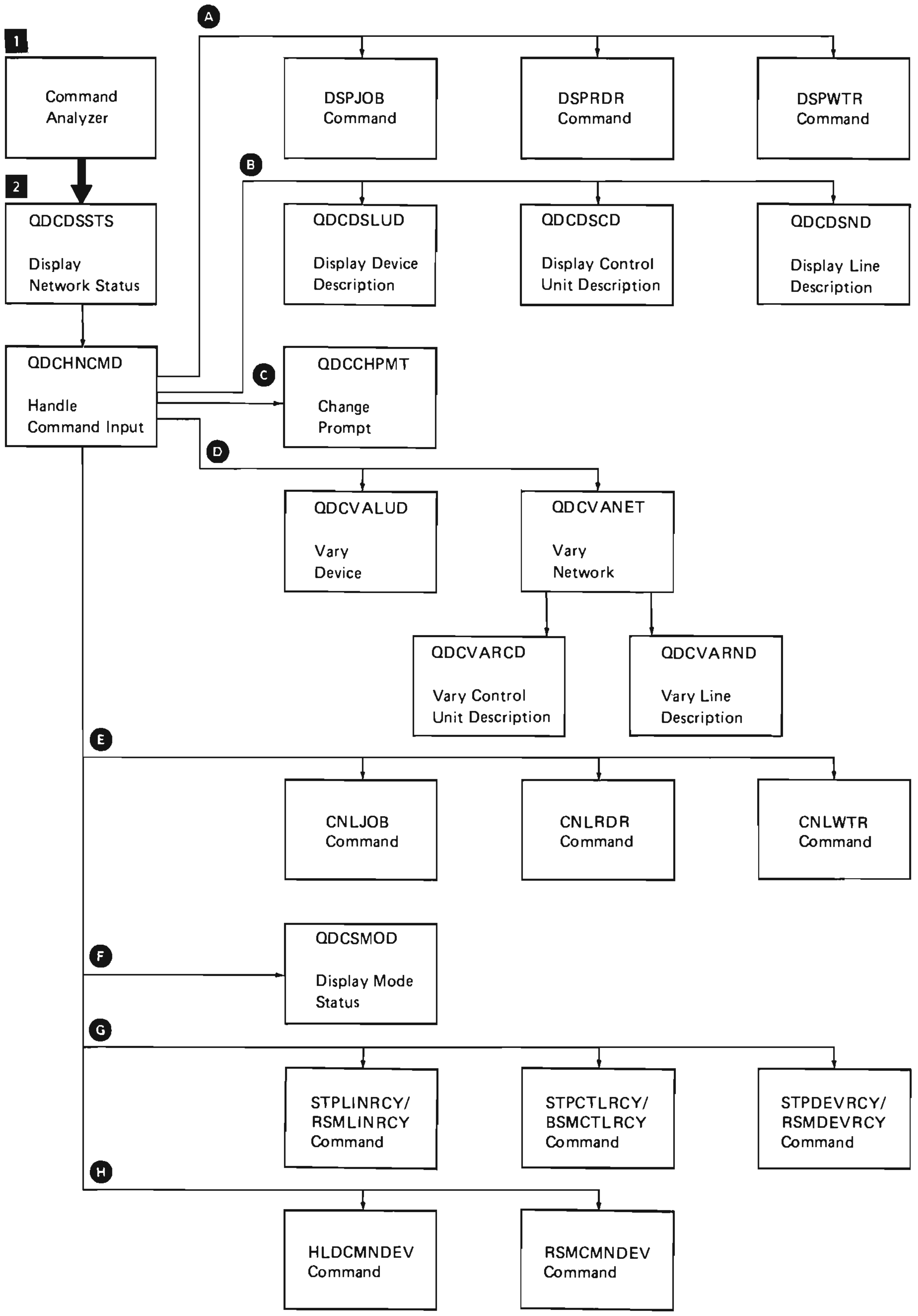

1
Command Analyzer
2
QDCDSSTS Display Network Status
QDCHNCMD Handle Command Input
A
DSPJOB Command
DSPRDR Command
DSPWTR Command
B
QDCDSLUD Display Device Description
QDCDSCD Display Control Unit Description
QDCDSND Display Line Description
C
QDCCHPMT Change Prompt
D
QDCVALUD Vary Device
QDCVANET Vary Network
QDCVARCD Vary Control Unit Description
QDCVARND Vary Line Description
E
CNLJOB Command
CNLRDR Command
CNLWTR Command
F
QDCSMOD Display Mode Status
G
STPLINRCY/ RSMLINRCY Command
STPCTLRCY/ BSMCTLRCY Command
STPDEVRCY/ RSMDEVRCY Command
H
HLDCMNDEV Command
RSMCMNDEV Command

Figure DC-8. Communication Status Display Commands Overview

Power and Vary Devices-Start CPF and Termination Procedures

Figure DC-9 shows an overview of the device configuration vary and power commands operation as well as the start CPF and termination procedures.

Start CPF Procedure

1 During start CPF, the system arbiter calls QDCINIT to control the status of line descriptions, control units, and devices.

- **E** A list of all NDs is obtained from the machine context. Each ND is checked for the auto vary flag being set on in the associated space. A list of all NDs with the vary flag on is sent to QDCVARND. QDCVARND varies those NDs online.
- **C** A list of all CDs is obtained. Each CD is checked for the power control feature and the auto vary flag. A list of CDs with the power control feature is sent to QDCPWCUD to be powered on.
- **D** A list of the CDs with the auto vary flag on is sent to QDCVARCD to be varied online.
- **A** A list of all LUDs is obtained. Each LUD is checked for the power control feature and auto vary flag. A list of the LUDs with the power control feature is sent to QDCPWLUD to be powered on.
- **B** A list of LUDs with the auto vary flag on is sent to QDCVALUD to be varied online.

Termination Procedure

2 QDCSHUTD provides the termination interface to control the status of line descriptions, control units, and devices.

- **B** A list of all LUDs is obtained from the machine context. Each LUD is checked for being online and having the power control feature. A list of all LUDs that are online is passed to QDCVALUD to be varied offline.
- **A** A list of all LUDs that have the power control feature is sent to QDCPWLUD to be powered off.
- **D** A list of all CDs is obtained and a list of those CDs that are online is passed to QDCVARCD to be varied offline.
- **C** A list of all CDs that have the power control feature and are powered on is sent to QDCPWCUD to be powered off.
- **E** A list of all NDs is obtained and a list of those NDs that are online is sent to QDCVARND to be varied offline.

Power Commands

3 The command analyzer decodes a Power Device (PWRDEV) command and control is transferred to QDCPWLUD.

- **G** QDCPWLUD passes a list of device names to QDCPRLUD, an event handler that executes under control of the system arbiter process, to change the power status of those devices that are specified in the status parameter of the Power Device command.

3 The command analyzer decodes a Power Control Unit (PWRCTLU) command and control is transferred to QDCPWCUD.

- **C** QDCPWCUD changes the power status of the control units with the power control feature as specified by the status parameter of the Power Control Unit command.

3 The command analyzer decodes a Vary Device (VRYDEV) command and control is transferred to QDCVALUD.

F For non-peer devices, QDCVALUD passes a list of device names to QDCVRLUD, an event handler under control of the system arbiter process, to change the online status of the devices as specified in the Vary Device command. If the device is varied offline, the lock on the LUD is passed to the system arbiter.

H For peer devices, an event is signaled to the logical unit services process to change the online status of the peer device, and to initiate initial session negotiation.

3 The command analyzer decodes a Vary Control Unit (VRYCTLU) command and control is transferred to QDCVARCD.

D QDCVARCD processes a list of control unit names to be varied online or offline as specified in the Vary Control Unit command.

3 The command analyzer decodes a Vary Line (VRYLIN) command and control is transferred to QDCVARND.

E QDCVARND processes a list of line description names to be varied online or offline as specified by the Vary Line Description command.

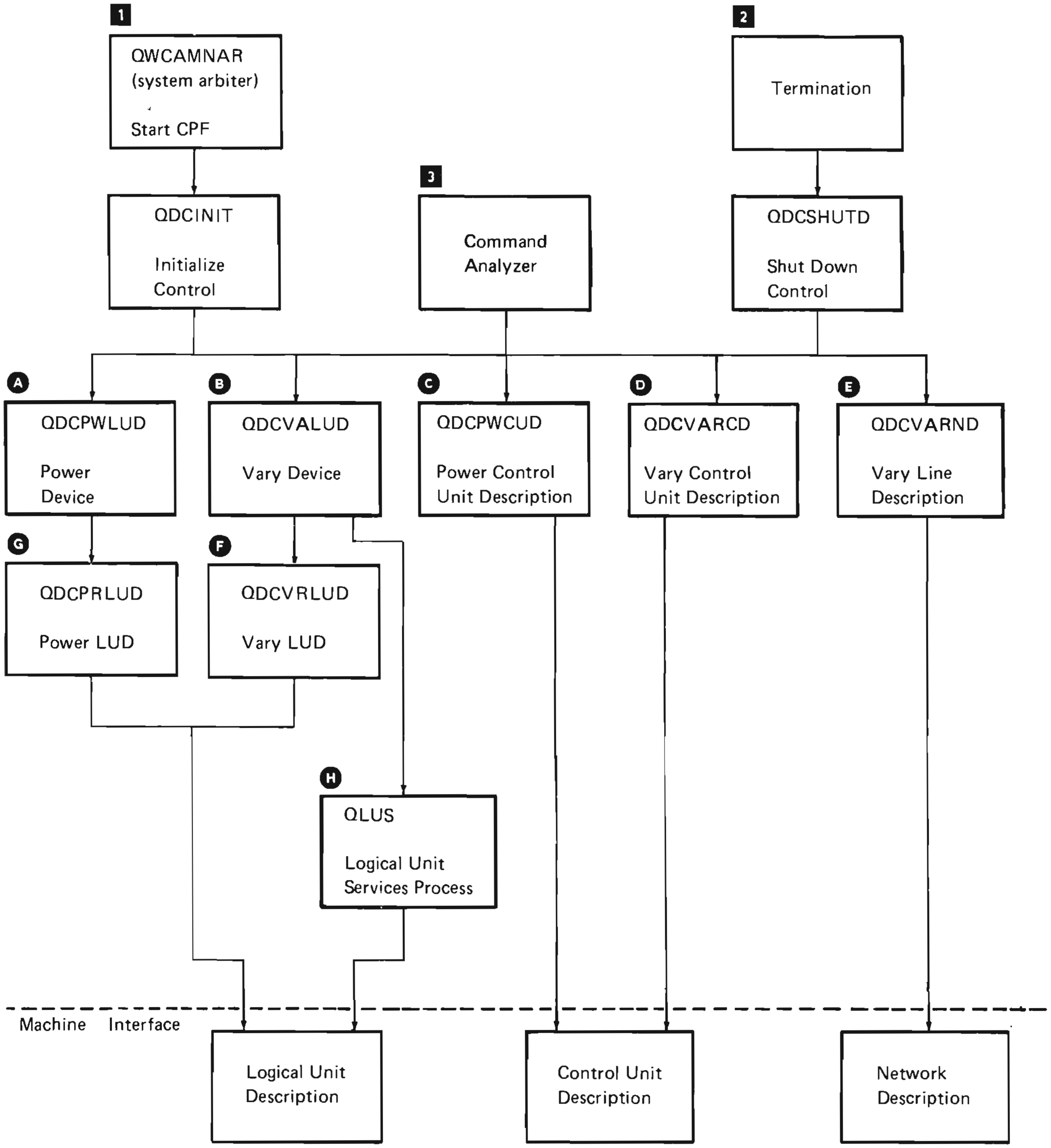

1
QWCAMNAR (system arbiter) Start CPF
2
Termination
3
Command Analyzer
QDCINIT Initialize Control
QDCSHUTD Shut Down Control
A
QDCPWLUD Power Device
B
QDCVALUD Vary Device
C
QDCPWCUD Power Control Unit Description
D
QDCVARCD Vary Control Unit Description
E
QDCVARND Vary Line Description
G
QDCPRLUD Power LUD
F
QDCVRLUD Vary LUD
H
QLUS Logical Unit Services Process
Machine Interface
Logical Unit Description
Control Unit Description
Network Description

Figure DC-9. Power and Vary Device Overview

Data Description

INTRODUCTION

The data description component of the CPF (control program facility) is the user interface to create, change, or delete the files and members for devices and the data base. In the data base, a physical and logical file can be created or deleted; physical file members and logical file members can be added or removed. Device files can be created, changed, or deleted for displays, the MFCU (multifunction card unit), printers, diskette, tape, save, communications (LU-1, peer, BSC) and mixed files.

The physical file, logical file, and device files for displays, printers, communications, and mixed files let the user enter a source description of the file using the data description specification forms. The source description is usually entered using the source entry utility, the copy component, or by spooled inline data. After the description is entered, a create command is used to process the source description and to create the specified type of file.

The data base file members and the MFCU, diskette, save, and tape device files do not use a source description. The parameters on the appropriate commands provide the information needed to create the file, add the member, or modify the device file.

GENERAL OVERVIEW

Data Description Modules

The data description component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

Device Related Modules

-->QDDCDPF–Create Device File (CRTDSPF, CRTPRTF)[1]: This module processes the create commands for display and printer.

-->QDDCDF–Create Device File (CRTCRDF, CRTSAVF, CRTDKTF, CRTTAPF)[1]: This module processes the create commands for MFCU, diskette, tape, and save.

-->QDDCCMF–Create Communications/BSC/Mixed File (CRTCMNF, CRTBSCF, CRTMXDF)[1]: This module processes the create command for secondary LU-1, peer, BSC, and mixed files. It also controls the invocation of other data description component modules used to process the required data description specifications.

> QDDREAD–See *Modules Related to Both Device and Data Base.*
>
> QDDSPRDV–Device File Syntax Processor: This module syntax processes the data description source and builds the IMS, which is used by QDDCDFDV and QDDPRINT.
>
> > QDDCKDV–Device File Syntax Checker: This module performs low-level syntax checking, one line at a time.
> >
> > QDDREFER–See *Modules Related to Both Device and Data Base.*

[1]This module is a CPP (command processing program).

QDDINIT–See *Modules Related to Both Device and Data Base.*

QDDCDFDV–Device IMS Processor: This module processes the intermediate source into another intermediate source format, input to the device file definition component, which is used to create the device file. Also, this module performs validity checking not performed by the syntax processor/checker.

QDDPRINT–See *Modules Related to Both Device and Data Base.*

-->QDDMDF–Change Device File (CHGDSPF, CHGSAVF, CHGPRTF, CHGCRDF, CHGDKTF, CHGTAPF)[1]: This module processes the change device file commands for display, printer, card, diskette device, save, and tape files.

-->QDDMCMF–Change Communications File (CHGCMNF,CHGBSCF, CHGMKDF)[1]: This module processes the Change Communications/BSC/Mixed File command.

-->QDDADDDV–Add Device File (ADDBSCDEVE, ADDCMNDEVE, ADDDSPDEVE)[1]: This module processes the mixed file add device entry commands for BSC, communications, and display devices.

-->QDDRMVDV–Remove Device File (RMVBSCDEVE, RMVCMNDEVE, RMVDSPDEVE)[1]: This module processes the mixed file remove device entry commands for BSC, communications, and display devices.

Data Base Related Modules

-->QDDCPF–Create Physical File (CRTPF and CRTSRCPF)[1]: This module is used to create a physical file from the Create Physical File and Create Source File commands and, optionally, using a specified source file containing additional descriptive information about the record format and the file.

QDDREAD–See *Modules Related to Both Device and Data Base.*

QDDSPRDB–Data Base File Syntax Processor: This module syntax processes the data description specification source for a data base file.

QDDCKDB–Data Base File Syntax Checker: This module performs the low-level syntax checking functions on a single line basis for a data base file.

QDDREFER–See *Modules Related to Both Device and Data Base.*

QDDINIT–See *Modules Related to Both Device and Data Base.*

QDDPFFLD–Physical File Field Processor: This module processes field descriptions for creation of a physical file record format.

QDDPRINT–See *Modules Related to Both Device and Data Base.*

-->QDDCPFM–Add Physical File Member (ADDPFM)[1]: This module is used to add a member to a physical file.

-->QDDCLF–Create Logical File (CRTLF)[1]: This module is used to create a logical file from a source description and the information in the call parameter list.

QDDREAD–See *Modules Related to Both Device and Data Base.*

QDDSPRDB–Data Base File Syntax Processor: This module syntax processes the data description specification source for a data base file.

QDDCKDB–Data Base File Syntax Checker: This module performs the low-level syntax checking functions on a single line basis for a data base file.

QDDREFER–See *Modules Related to Both Device and Data Base.*

QDDINIT–See *Modules Related to Both Device and Data Base.*

QDDLFFLD–Logical File Field Processor: This module processes field descriptions for creation of a logical file record format.

[1]This module is a CPP (command processing program).

QDDPRINT–See *Modules Related to Both Device and Data Base.*

-->QDDCLFM–Add Logical File Member (ADDLFM)[1]: This module is used to add a member to a logical file.

-->QDDDMBR–Remove Member (RMVM)[1]: This module is used to delete physical or logical data base file members.

Modules Related to Both Device and Data Base

-->QDDREAD–Common DDS Read: This module contains the function necessary to read the DDS into a source space (102-byte records) for processing by the syntax processor and QDDPRINT. It also performs some initialization functions such as space and index creation, and opening and closing the source input file.

-->QDDPRINT–Common DDS Print: This module contains the function necessary to print the DDS source listing, the expanded source listing, and the error summary. The original source listing is derived from the information read (QDDREAD) and from error information accumulated while the source was being processed. The expanded source listing is generated from the IMS built by the syntax processors. The error summary shows the identifier number of the diagnostic message, its severity, and its text. A completion message is also generated. When the file is not created, additional messages will exist with explanations as to why the file was not created.

-->QDDINIT–Keyword Table Initialization: This module is invoked to store the appropriate keyword table in the syntax processor's program associated space. This reduces the initialization time required to invoke the syntax processor, and is done only once per installation.

-->QDDESPEH–Extend Space Exception Handler: This module is used to handle the MCH0601 (space addressing violation) exception. It increases the size of the space object and then returns to retry the instruction causing the exception.

-->QDDREFER–Field Reference Processor: This module processes reference-related specifications. It extracts field reference information from a data base file or the current source, and is called from the syntax processor.

DDS Single Line Syntax Checker

-->QDDSEU–Single Line Syntax Checker Bridge Module: This module is used to invoke the appropriate checker:

- Check Physical File Description: The module interfaces with QDDCKDB.
- Check Logical File Description: The module interfaces with QDDCKDB.
- Check Display File Description: The module interfaces with QDDCKDV.
- Check Printer File Description: The module interfaces with QDDCKDV.
- Check Communications/BSC File Description: The module interfaces with QDDCKDV.
- Mixed File Description: The module interfaces with QDDCKDV.

QDDINIT–See *Modules Related to Both Device and Data Base.*

Creating Files With a Source Description Provided

Data description has a separate call interface to create a physical file, logical file, display file, printer file, communications file, or BSC file. These calls are generated from the corresponding create command. The command analyzer checks the create command parameters for errors. If no errors are found, the parameters are passed to data description using its call interface. Data description then performs additional error checking on the create command parameters. If any errors are found, diagnostic messages are issued and an escape message is sent.

Common data management is invoked to open the file containing the source description and to get each record in the file. Data description scans each record for errors; if any errors are found, an error indication line is generated to be printed on the source listing. An internal form of the source description is also generated to be passed to data base or device file definition to create the file.

[1]This module is a CPP (command processing program).

In addition to the source listing, a second listing is produced that shows any defaults that were made, to let processing continue and also to show any retrieved information from referenced field descriptions that were used to create the file.

The error messages and error statistics are printed on the source listing.

Data management is called, which then invokes data base or device file definition to create the file from the internal form created by data description. If data management, data base, or device file definition detects any additional errors, an escape message is sent to data description.

Creating a File or Adding a File Member Without Supplying a Source Description

Physical files, physical file members, logical file members, and device files (excluding communications, mixed files, and binary synchronous communications files) can be created without supplying a source description by using the call interface to data description. The calls are generated by using the corresponding create file command or add member command. The command analyzer checks the command parameters for errors. If no errors are found, the command analyzer passes the parameters to the data description component.

Data description performs additional error checking and generates appropriate input for data base or device file definition. Common data management is then called to create the file or add the file members. If any errors are detected by either common data management, data base definition, device file definition, or data description, an escape message is sent.

Changing Device Files

Data description can change device files by using a change file command. The command analyzer checks the command parameters for errors. If no errors are found, the change parameters are passed to the data description component. Data description performs additional error checks on the parameters and calls common data management, which invokes device file definition to modify the device file. If any errors are detected by either common data management or data description, an escape message is sent.

Single Line Syntax Checking Through Source Entry Utility

The data description component syntax checks single lines of the data description source specifications as they are being entered. A routine is provided to syntax check each of the types of source specifications: physical file, logical file, printer device file, communications device file, BSC device file, mixed device file, and display device file. If errors are detected, they are returned to the caller in the form of a message string containing message IDs and any replacement text.

Multiple Line Syntax Checking Through Screen Design Aid

The data description component provides a callable routine to syntax check multiple lines of display data description source specifications. The parsed output provided by the data description component is returned to the caller.

This page is intentionally left blank.

Create Device File Overview

Figure DD-1 and the following text describe how a device file is created.

1 The command analyzer transfers control to QDDCDF, QDDCCMF, or QDDCDPF. It is passed pointers to command parameters that were entered on the Create Display File, Create Printer File, Create Diskette File, Create Card File, Create Save File, Create Tape File, or Create Communications/BSC/Mixed Device File command. The file attributes entered on the command are placed in a create-input space.

2 If the file to be created is a diskette, card, save, or tape file, or if it is a display or printer device file *without* a source description, the ?CRTDEVF macro calls the QDMROUTE module of common data management, which invokes device file definition to create the device file. A pointer to a structure that contains the qualified name of the file being created and a pointer to the space that contains the input for the new file are passed to QDMROUTE.

3 If the Create command also specifies a DDS source file (required for communications, BSC, and mixed files), then QDDREAD is called to open the source file and read the source records. QDDREAD also provides other initialization functions, such as space and index creation.

D QDDESPEH is called to extend spaces as the result of handling an out-of-space exception.

4 QDDSPRDV (see Figure DD-1) is called once to syntax check the DDS source specification. QDDSPRDV (see Figure DD-1) is passed a pointer to a data structure that contains addressability to: the source input space, intermediate source space, and error summary space.

A QDDINIT is called to initialize the device file keyword table (once per install).

B QDDREFER is called to retrieve field description information from a previously defined field in a record format in the data base or from the current source. QDDREFER is passed a pointer to a data structure that contains addressability to the intermediate source space, the source error index, and work spaces so that field information can be extracted. The ?EXTFILED macro in QDDREFER calls QDMROUTE, which invokes data base definition to extract the field description from the data base file.

C QDDCKDV (see Figure DD-1) is called by QDDSPRDV for each logical line of DDS source for the file.

D QDDESPEH is called to extend spaces as the result of handling an out-of-space exception.

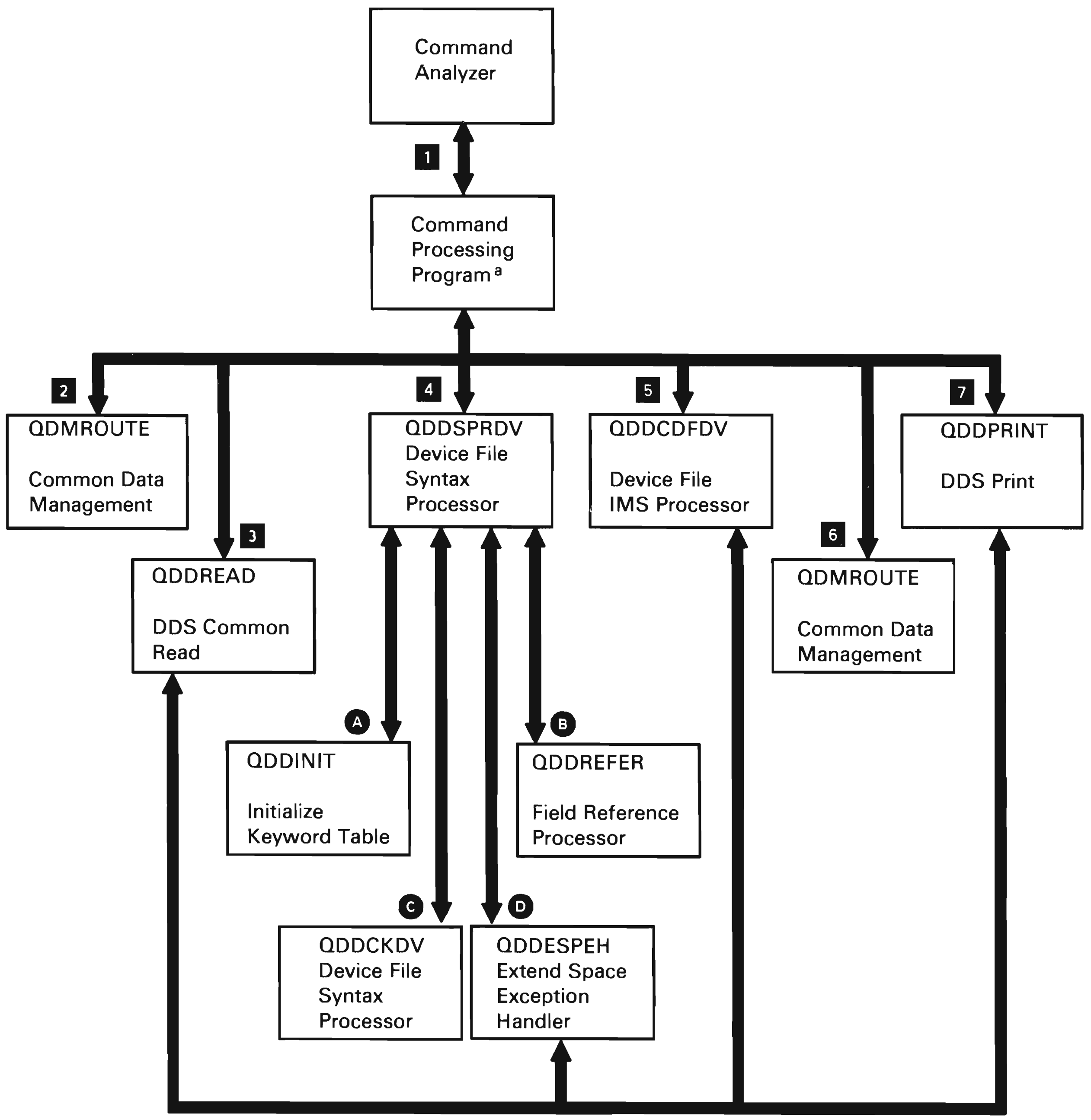
5 QDDCDFDV (see Figure DD-1) is called to process the intermediate space. Further syntax processing takes place (location processing) from the IMS, and the created input space is updated to contain information to be used to create the device file.

D QDDESPEH is called to extend spaces as the result of handling an out-of-space exception.

6 If the file to be created is a display or printer device file with a source description, or a communications, BSC, or mixed file, the ?CRTDEVF macro calls the QDMROUTE module of common data management, which invokes device file definition to create the device file. A pointer to a structure that contains the qualified name of the file being created and a pointer to the space that contains the input for the new file are passed to QDMROUTE.

7 QDDPRINT is called to open the DDS printer file (QPDDSSRC) to print the source listings (original source with diagnostics and expanded source showing defaults, external/source references, and buffer positions) and the error summary.

D QDDESPEH is called to extend spaces as the result of handling an out-of-space exception.


Command Analyzer
1
Command Processing Program[a]
2
QDMROUTE Common Data Management
3
QDDREAD DDS Common Read
4
QDDSPRDV Device File Syntax Processor
5
QDDCDFDV Device File IMS Processor
6
QDMROUTE Common Data Management
7
QDDPRINT DDS Print
A
QDDINIT Initialize Keyword Table
B
QDDREFER Field Reference Processor
C
QDDCKDV Device File Syntax Processor
D
QDDESPEH Extend Space Exception Handler

[a] QDDCDF for card, tape, save, and diskette; QDDCCMF for communications (LU-1, peer, BSC), and mixed file; QDDCDPF for display and printer.

PAAB042-0

Figure DD-1. Create Device File Overview

Create Physical File/Add Physical File Member Overview

Figure DD-2 and the following text describe the create physical file, create source file, and add physical file member functions.

Create Physical File/Create Source File

1 The command analyzer transfers control to QDDCPF. It is passed pointers to the physical file attributes that were entered as command parameters. If a record length is specified on the Create Physical File (CRTPF) command (defaulted on the Create Source Physical File command), a record format is generated using the record length. Control is then passed to the QDMROUTE module of common data management (see **E**). If, however, a source file is specified on the Create Physical File command, that file is opened, read, and processed.

A QDDREAD is called to open the source file and read the source records. It also provides other initialization functions, such as space and index creation.

B QDDSPRDB is called to syntax check the DDS source specifications. QDDSPRDB is passed a pointer to a data structure that contains addressability to a space for the new record format definitions, the intermediate source space, the error summary space, and so forth. QDDCKDB, the data base file syntax checker, is called by QDDSPRDB for each logical line of DDS source for a physical file.

QDDREFER is called to retrieve field description information from a previously defined field in a record format in the data base or from the current source. QDDREFER is passed a pointer to a data structure that contains addressability to the intermediate source space, the source error index, and work spaces so that field information can be extracted. The ?EXTFILED macro in QDDREFER calls QDMROUTE, which invokes data base definition to extract the field description from the data base file.

QDDINIT is called to initialize the data base file keyword table (once for install).

C QDDPFFLD is called to process all of the field specifications for a new record format. It is passed a pointer to a data structure that contains addressability to a space for the new record, to the error summary space, and so forth.

D If an out-of-space exception is signaled, QDDESPEH is called to extend the space.

E The ?CRTDBF macro in QDDCPF is used to call the QDMROUTE module of common data management, which invokes data base definition to create the new physical file. QDMROUTE is passed a pointer to a data structure that contains the name of the file to be created as well as a pointer to the file definition template.

F QDDPRINT is called to open the printer file (QPDDSSRC) to print the source listings (original source with diagnostics and expanded source, showing defaults, references, and buffer positions) and the error summary.

2 If the physical file is successfully created and the Create Physical File or Create Source File commands indicate that a physical file member is also to be added, QDDCPFM is called and pointers are passed to the appropriate command parameters (see **3**).

Add Physical File Member

3 The command analyzer and QDDCPF transfer control to QDDCPFM. It is passed pointers to the physical file member attributes that were entered as command parameters.

G The ?CRTDBM macro in QDDCPFM is used to call the QDMROUTE module of common data management, which invokes data base definition to add a member to the physical file. QDMROUTE is passed a pointer to a data structure that contains the file name, member name, and a pointer to the member definition template.

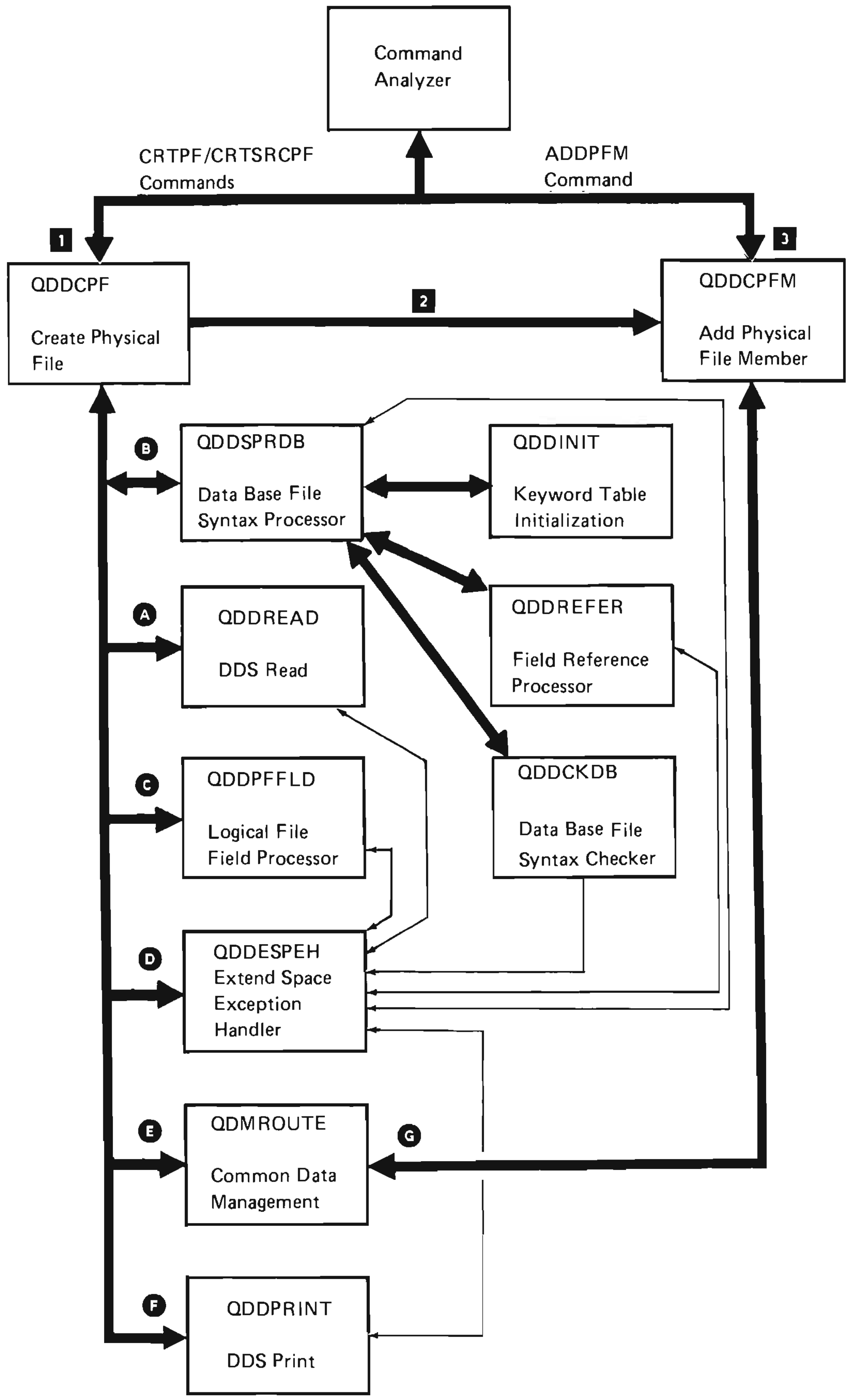

Command Analyzer
CRTPF/CRTSRCPF Commands
ADDPFM Command
1
2
3
QDDCPF Create Physical File
QDDCPFM Add Physical File Member
B
QDDSPRDB Data Base File Syntax Processor
QDDINIT Keyword Table Initialization
A
QDDREAD DDS Read
QDDREFER Field Reference Processor
C
QDDPFFLD Logical File Field Processor
QDDCKDB Data Base File Syntax Checker
D
QDDESPEH Extend Space Exception Handler
E
QDMROUTE Common Data Management
G
F
QDDPRINT DDS Print

Figure DD-2. Create Physical File/Create Source File/Add Physical File Member Overview

Create Logical File/Add Logical File Member Overview

Figure DD-3 and the following text describe the create logical file and add logical file member functions.

Create Logical File

1 The command analyzer transfers control to QDDCLF. It is passed pointers to the file attributes that were entered as command parameters.

- **A** See *Create Physical File/Create Source File.*

- **B** QDDSPRDB is called, to syntax check the DDS source specifications. QDDSPRDB is passed a pointer to a data structure that contains addressability to the source input space, intermediate source space, error summary space, and so forth.

  QDDCKDB is called by QDDSPRDB for each logical line of DDS source for a logical file.

  QDDINIT is called to initialize the data base file keyword table (once per install).

- **C** QDDLFFLD is called to process field specifications for a new record format, and is passed a pointer to a data structure that contains addressability to a space for new record format descriptions, intermediate source data, error summary space, and so forth.

- **D** If an out-of-space exception is signaled, QDDESPEH is called to extend the space.

- **E** The ?CRTDBF macro in QDDCLF is used to call the QDMROUTE module of common data management, which invokes data base definition to create the logical file. QDMROUTE is passed a pointer to a data structure that contains the name of the logical file to be created and a pointer to the file definition template.

- **F** See *Create Physical File/Create Source File.*

2 If the logical file is successfully created and the Create Logical File (CRTLF) command indicates that a logical file member is also to be added, QDDCLF transfers control to QDDCLFM. It is passed pointers to the appropriate command parameters (see **3**).

Add Logical File Member

3 The command analyzer and QDDCLF transfer control to QDDCLFM. It is passed pointers to the logical file member attributes that were entered as command parameters.

- **G** The ?EXTFILED macro in QDDCLFM is used to call the QDMROUTE module of common data management, which invokes data base definition to extract file definitions.

The ?CRTDBM macro in QDDCLFM is used to call the QDMROUTE module of common data management, which invokes data base definition to add a member to the logical file. QDMROUTE is passed a pointer to a data structure that contains the file name and member name, as well as a pointer to the member definition template.

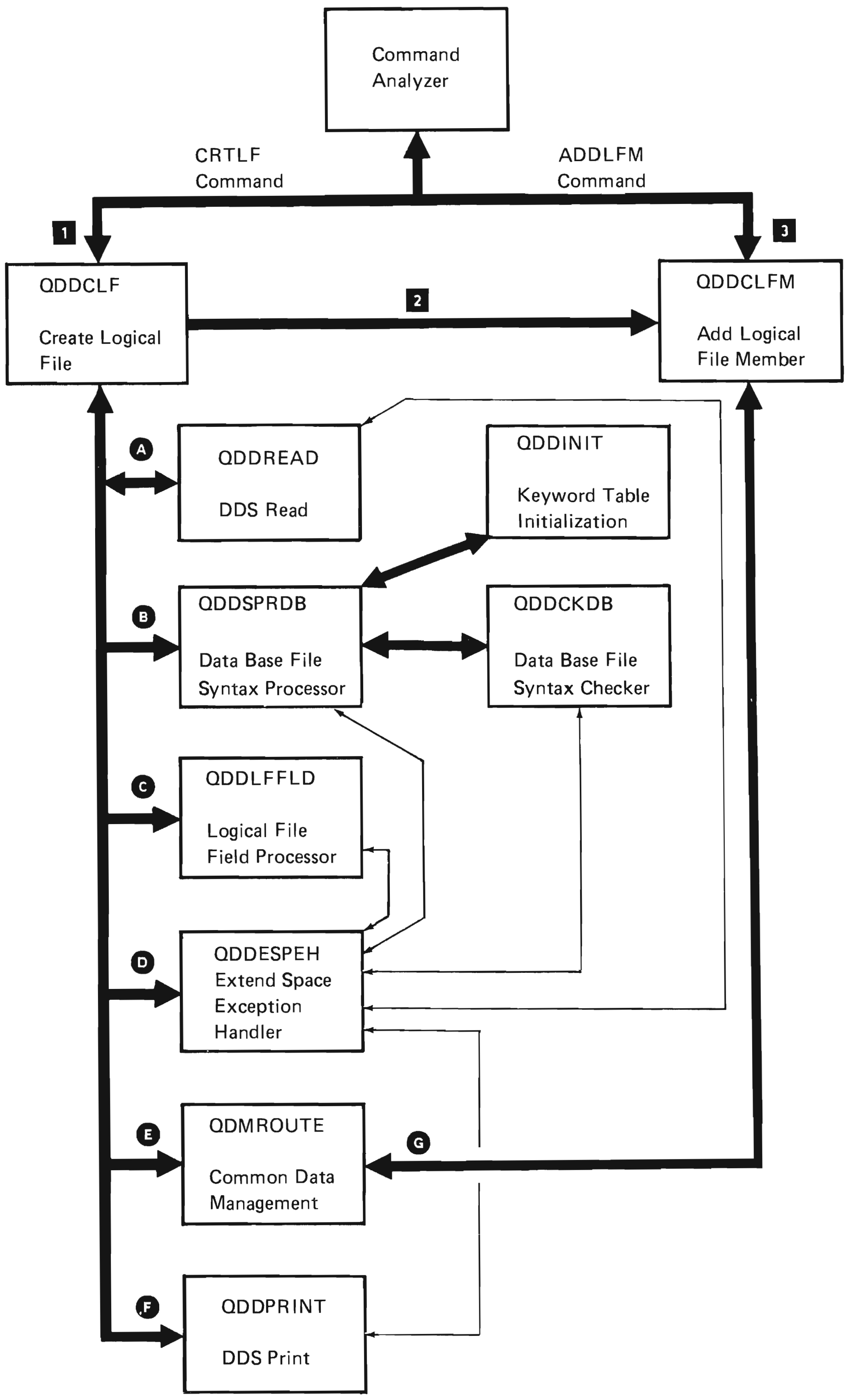


Figure DD-3. Create Logical File/Add Logical File Member Overview

Change Device File/Remove Member Overview

Figure DD-4 and the following text describe the change device file and remove member functions.

Change Device File

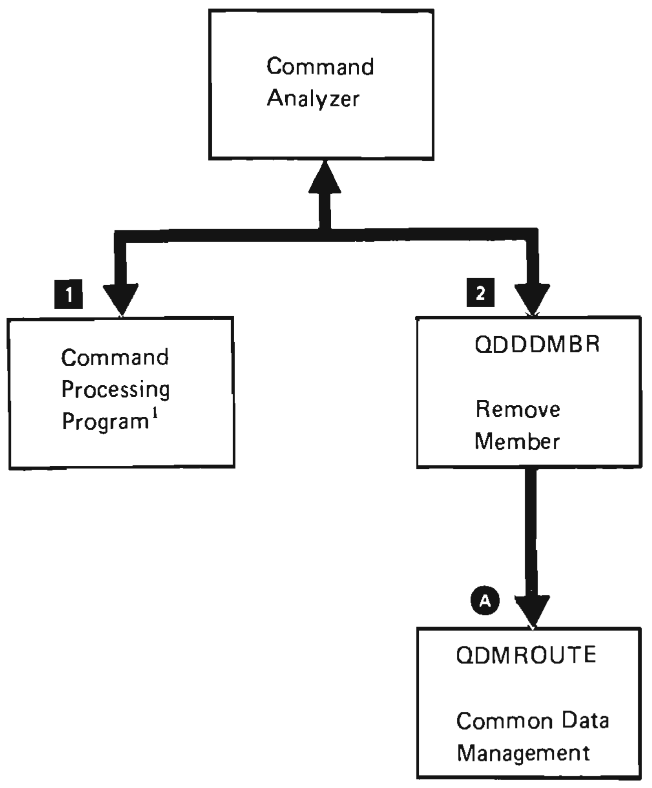
1 The command analyzer transfers control to the command processing program. It is passed pointers to the file attributes entered on the Change Display File, Change Printer File, Change Diskette File, Change Card File, Change Tape File, Change Communications File, Change BSC File, Change Mixed File, or Change Save File command.

- **A** The ?MDFDEVF macro in QDDMDF or QDDMCMF is used to call device file definition to change the device file.

Remove File Member

2 The command analyzer transfers control to QDDDMBR. It is passed a pointer to the file and member name that is to be deleted. In the case of a *GENERIC remove, an alphabetical list of names is retrieved, using the EXTFILED macro. The DLTDBM macro is invoked for each member name that meets the generic name criteria.

- **A** The ?DLTDBM macro in QDDDMBR is used to call the QDMROUTE module of common data management, which invokes data base to remove the member from the file. QDMROUTE is passed a pointer to a data structure that contains the qualified name of the member that is to be removed.



[1] QDDMDF handles the CHGDSPF, CHGPRTF, CHGDKTF, CHGSAVF, CHGCRDF, and CHGTAPF commands. QDDMCMF handles the CHGCMNF, CHGBSCF and CHGMXDF commands.

Figure DD-4. Change Device File/Remove Member Overview

Single Line Syntax Checker Overview

Figure DD-5 and the following text describe the syntax check file description function.

1 The caller specifies the type of single line syntax check desired via the ?CALLDDS macro. The caller passes a logical source line. Returned to the source entry utility are:

- A return code that indicates if any errors were found.
- A message stack for the errors, if any, that were detected.
- A string, highlighted on the display, indicating the fixed columns that were in error.

The ?CALLDDS macro invokes the module QDDSEU and, based upon the type of checker desired, one of the following single line syntax checkers is invoked with a transfer of control.

A QDDCKDB is called to perform the actual syntax checking functions for physical file and logical file DDS source.

B QDDCKDV is called to perform the actual syntax checking functions for device file DDS source.

2 QDDINIT–The QDDINIT module stores the device and data base keyword tables in the program-associated space of QDDSEU.

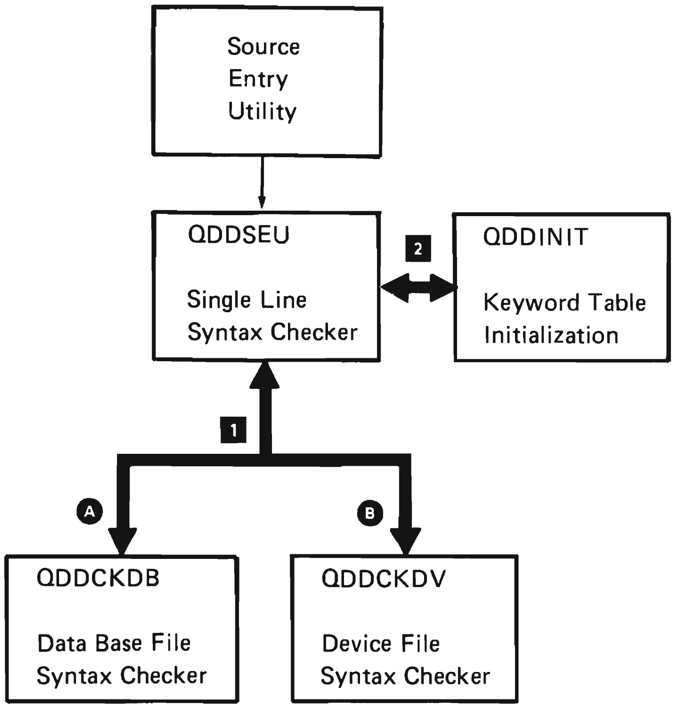

Source Entry Utility
QDDSEU
Single Line Syntax Checker
2
QDDINIT
Keyword Table Initialization
1
A
B
QDDCKDB
Data Base File Syntax Checker
QDDCKDV
Device File Syntax Checker

Figure DD-5. Syntax Check File Description Overview

Screen Design Aid DDS Parser

Figure DD-6 and the following text describe the screen design aid DDS parser function.

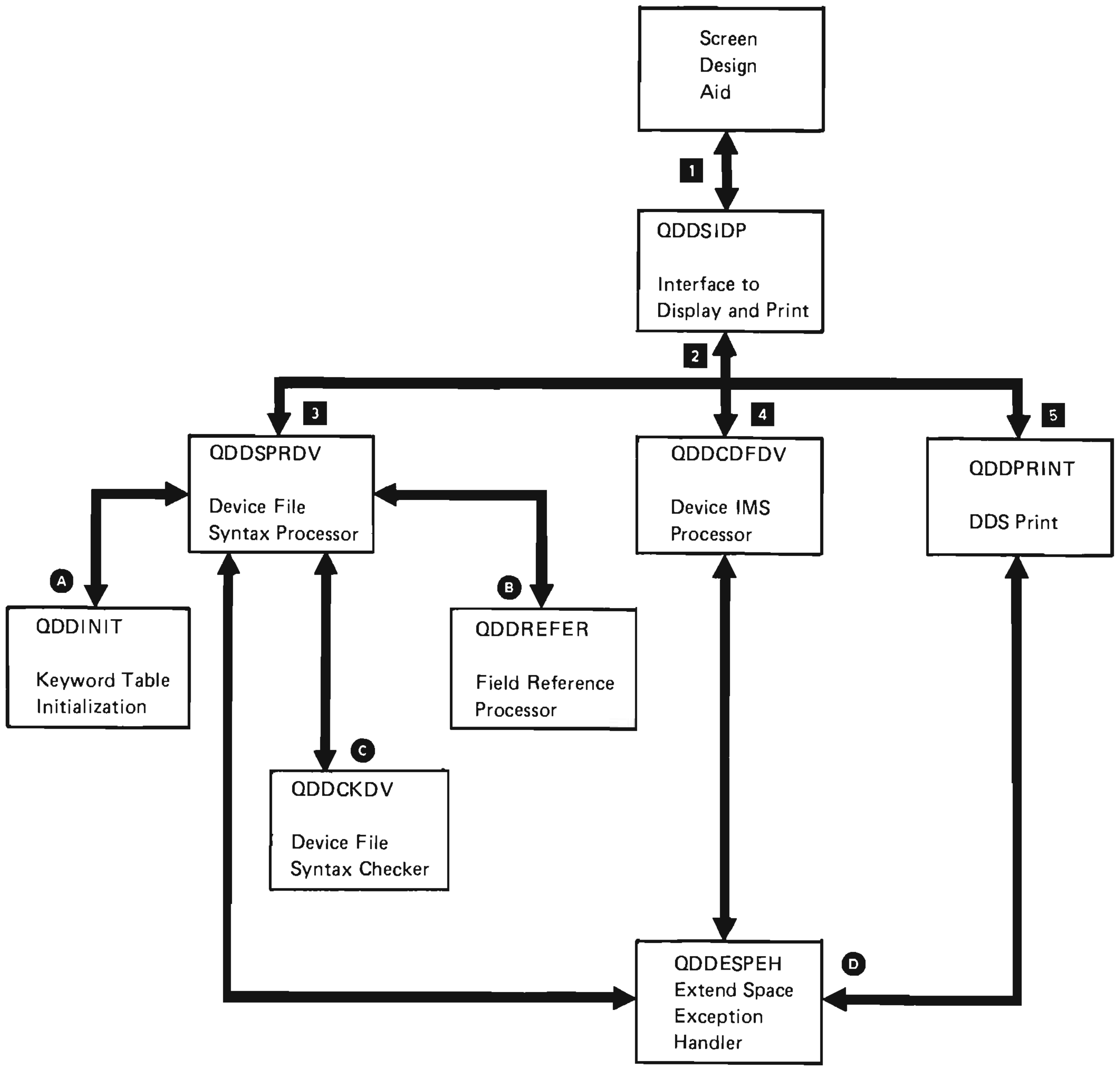
1 The caller specifies if printing is desired via the ?CALLDDSI macro, along with other required information. This operation allows the caller to block an entire set of DDS source into one space, providing the caller has described the space by specifying the number of records and the length of each (92 to 102 characters).

The information returned is a parsed space described by the ?WWDFDDSI macro.

The screen design aid uses this interface to ignore invalid DDS and build from the parsed output back to the source file.

2 The module QDDSIDP is invoked via the ?CALLDDSI macro. This module emulates the DDS create display command processing program and the QDDREAD module function.

QDDSIDP provides an interface to the function described in **4**, **5**, and **7** of Figure DD-1.


Screen Design Aid
1
QDDSIDP
Interface to Display and Print
2
3
QDDSPRDV
Device File Syntax Processor
4
QDDCDFDV
Device IMS Processor
5
QDDPRINT
DDS Print
A
QDDINIT
Keyword Table Initialization
B
QDDREFER
Field Reference Processor
C
QDDCKDV
Device File Syntax Checker
D
QDDESPEH
Extend Space Exception Handler

Note: See Figure DD-1 4, 5, and 7 for a description of 3, 4, and 5 above.

Figure DD-6. Screen Design Aid DDS Parser Overview

INTRODUCTION

The device file definition component of the CPF (control program facility) creates, changes, deletes, and extracts information from device files. It also retrieves the current release/modification level for device files, and if necessary, converts device files to the current release/modification level. Device file definition also provides the following subset of operations:

- Move device files from one library to another library
- Retrieve the size of a specific device file
- Rename a device file
- Grant authority to a device file
- Revoke authority to a device file
- Transfer ownership of a device file
- Add and remove device entries from a mixed file
- Save/restore an online save file
- Create a duplicate online save file
- Dump a device file

GENERAL OVERVIEW

Device File Definition Modules

The device file definition component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QDFCDF–Create Device File: This module creates a device file.

QDFVDLST–Validate Device List: This module ensures that the device descriptions match the device file type.

QDFBLDDF–Build Device File: This module controls the building of the device file.

QDFDFTPR–Build Nonfield-Level Device File: This module controls the building of device files that are created without data description specifications.

QDFLVLGN–Level Check Generator: This module generates the level check values for the device file.

QDFCDFPR–Create Printer Device File: This module processes the record formats for printer files created with data description specifications.

QDFRCDPR–Record Processor: This module processes the record formats for display, communications, and mixed files created with data description specifications.

QDFKWDPR–Keyword Processor: This module processes the keywords specified through the data description specifications for device files.

QDFCRTWU–Create Where-Used Section: This module creates the extract and where-used section for device files.

-->QDFMDF–Modify Device File: This module changes the attributes of a device file.

QDFVDLST–Validate Device List: This module ensures that the device descriptions match the device file type.

-->QDFDDF–Delete Device File: This module deletes the device file.

-->QDFEDF–Extract Device File: This module extracts file attributes, the format name list, field descriptions, and record formats from a device file.

-->QDFCNVPP–Retrieve/Update Current Release/Modification Level: This module either retrieves or updates the release/modification level used for creating and converting files.

-->QDFMOVE–Move Device File: This module transfers addressability from one library to another library.

-->QDFRENAM–Rename Device File: This module changes the name of a device file.

-->QDFSIZE–Device File Size: This module retrieves the size of a device file for the Display Object Description command.

-->QDFGRANT–Grant Authority: This module is used to grant authority to a device file.

-->QDFREVOK–Revoke Authority: This module is used to revoke the authority of a user to a device file.

QDFCHKFL–Check File Status: This module is called by QDFMOVE, QDFRENAM, QDFDDF, QDFSIZE, QDFRSTDF, QDFDUPDF, and QDFMDF to handle the status checking of the device file as well as locking and unlocking.

-->QDFXOWNR–Transfer Ownership: This module transfers device file ownership from one owner to another owner.

-->QDFCVALL–Convert Files: This module controls the conversion of files that are not at the current release/modification level.

QDFCNVPP–Retrieve/Update Current File Release/Modification Level: This module retrieves and updates the release/modification level used for file creation and file conversion.

-->QDFCNVF–Convert Device File: This module converts a device file to the current release/modification level.

QDFMATR1–Materialize Display and Printer Files: This module materializes display and printer files to provide source for file creation process.

QDFMATR2–Materialize Communications and BSC Files: This module materializes communications and BSC files to provide source for the file creation process.

QDFBLDDF–Build Device File: This module controls the building of the device file.

QDFDFTPR–Build Nonfield-Level Device File: This module controls the building of device files that are created without data description specifications.

QDFLVLGN–Level Check Generator: This module generates the level check values for the device file.

QDFCDFPR–Create Printer Device File: This module processes the record formats for printer files created with data description specifications.

QDFRCDPR–Record Processor: This module processes the record formats for display, communications, and mixed files created with data description specifications.

QDFKWDPR–Keyword Processor: This module processes the keywords specified through the data description specifications for device files.

QDFCRTWU–Create Where-Used Section: This module creates the extract and where-used section for device files.

-->QDFDMPDF–Dump Device File: This module dumps a device file. For an online save file, it dumps the dump space also.

-->QDFDUPDF–Duplicate Device File: This module creates a duplicate of an online save file. The contents of the dump space are not duplicated.

-->QDFSAVDF–Save Device File: This module saves the description of an online save file. The contents of the dump space are not saved.

-->QDFRSTDF–Restore Device File: This module restores an online save file. If the file does not already exist on the system, an empty dump space is created.

Device File Definition Overview

Figure DF-1 shows an overview of the relationship of other CPF components to the device file definition component.

1 Common data management is invoked during create and change to build the prototype ODP (open data path).

2 Data description: The data description component is the primary user of device file definition. It uses the device file definition component to complete the creating, and changing of device files.

3 Librarian: The librarian component uses the device file definition component to delete, move, and rename files. It also is used to retrieve the size of device files and to create a duplicate of an online save file.

4 Security: The security component uses the device file definition component to grant and revoke authority for device files and to transfer ownership of device files.

5 High-level languages and utilities: They use the device file definition to extract, create, change, and delete device files.

6 Service: The service component is used by the device file definition component to build the service information stored with the device file object.

7 Common data management invokes device file definition to retrieve the current release/modification level for device files.

8 Save/restore, reclaim/damage notification, and installation invoke device file definition to determine if a device file is at the current release/modification level, and if not, device file definition will convert the device file when necessary. Save/restore also invokes device file definition to handle the save and restore of online save files.

9 Service invokes device file definition to dump the device file.

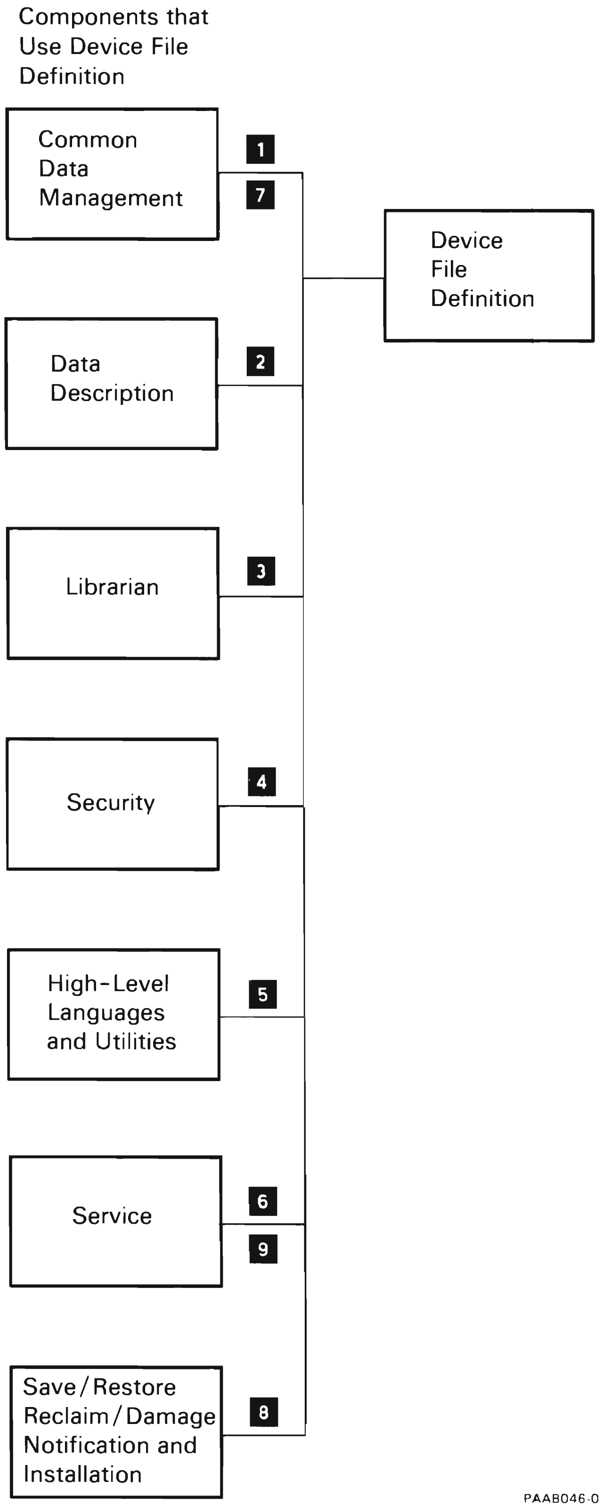

Components that Use Device File Definition
Common Data Management
1
7
Device File Definition
Data Description
2
Librarian
3
Security
4
High-Level Languages and Utilities
5
Service
6
9
Save/Restore Reclaim/Damage Notification and Installation
8
PAAB046-0

Figure DF-1. Device File Description Overview

Create Device File Definition

Figure DF-2 and the following text describe the device file definition create operation.

1 QDMROUTE is called by the ?CRTDEVF macro. A parameter list is passed that contains pointers to the file attributes and if a field level file is being created, the record formats and file-level keyword information. The qualified name of the file to be created is also passed.

2 When a device file is to be created, QDMROUTE transfers control to QDFCDF to create the file.

3 QDFCDF calls QDFBLDDF to build the device file.

4 If any device descriptions are associated with this file, QDFCDF call QDFVDLST to validate the list.

5 Control is returned to the caller.

6 Common data management is invoked to initialize the prototype ODP (open data path) contained in every device file.

7 Librarian is invoked to assign the authority of the file and build the OIR (object information repository) information.

8 Service is invoked to build the service information stored in the OIR for a device file.

9 QDFBLDDF calls QDFLVLGN to generate the level check values for the device file.

10 QDFBLDDF calls QDFRCDPR to process the record formats.

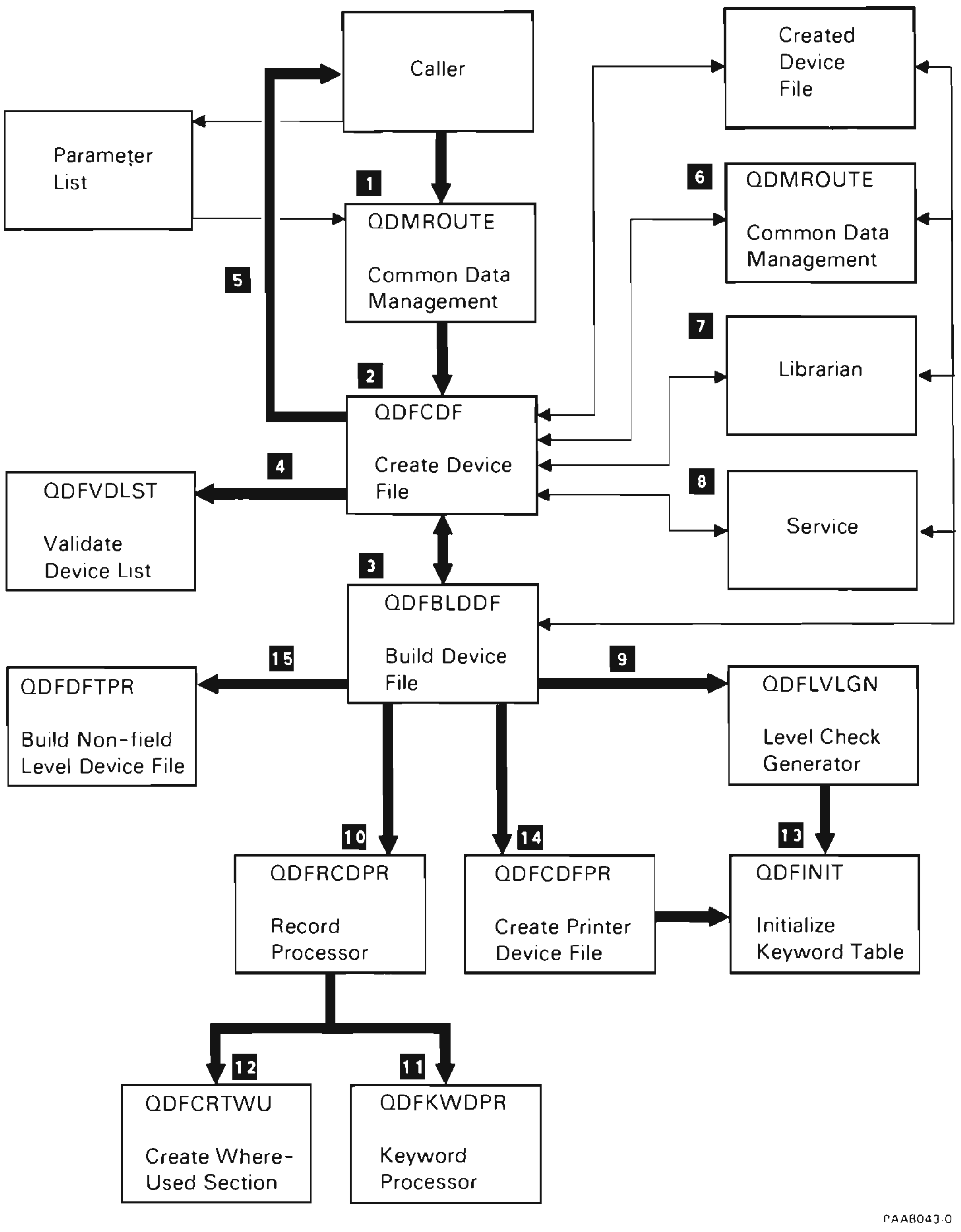
11 QDFRCDPR calls QDFKWDPR to process any keywords specified through the data description specifications.

12 QDFRCDPR calls QDFCRTWU to create the where-used section used by extract.

13 QDFLVLGN calls QDFINIT to initialize keyword tables.

14 QDFBLDDF calls QDFCDFPR to create and process the record formats, fields, and keywords for the externally described printer files. QDFCDFPR also builds the where-used section used by the extract function.

15 QDFBLDDF calls QDFDFTPR to build files that are not externally described.



Caller
Parameter List
1
QDMROUTE Common Data Management
5
2
QDFCDF Create Device File
4
QDFVDLST Validate Device List
3
QDFBLDDF Build Device File
15
QDFDFTPR Build Non-field Level Device File
9
QDFLVLGN Level Check Generator
10
14
13
QDFRCDPR Record Processor
QDFCDFPR Create Printer Device File
QDFINIT Initialize Keyword Table
12
11
QDFCRTWU Create Where-Used Section
QDFKWDPR Keyword Processor
Created Device File
6
QDMROUTE Common Data Management
7
Librarian
8
Service
PAAB043-0

Figure DF-2. Create Device File Definition Overview

Change Device File Definition

Figure DF-3 and the following text describe the device file definition change operation.

1 QDFMDF is called by the ?MDFDEVF macro. A parameter list is passed that contains a pointer to the modify information and the qualified name of the device file being changed.

2 Control is returned to the caller.

3 Common data management is invoked to build a new prototype ODP.

4 Librarian is invoked to change OIR information.

5 QDFMDF calls QDFCHKFL to check authority and lock the object involved in the change device file.

6 QDFMDF calls QDFVDLST to ensure the device descriptions specified for this file are valid.

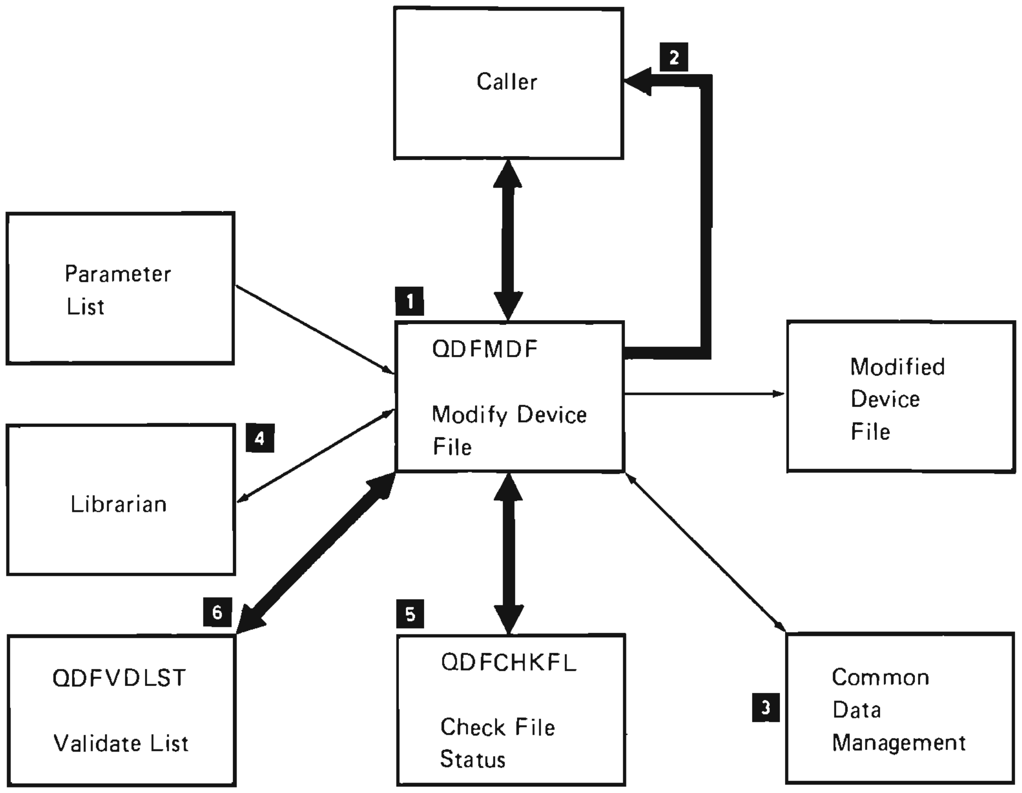


Figure DF-3. Change Device File Definition Overview

Delete Device File Definition

Figure DF-4 and the following text describe a delete device file definition operation.

1 The librarian or some other function calls the QDMROUTE module of common data management. A parameter list is passed that contains a pointer to the qualified name of the file being deleted.

2 If a device file is to be deleted, QDMROUTE calls QDFDDF to delete the device file. A resolved system pointer to the file is passed to QDFDDF.

3 QDFDDF calls QDFCHKFL to check file status and lock the object involved in the delete.

4 The ?DLTOIR macro is issued to invoke the librarian component to delete the OIR entry and update the library.

5 Control is returned to the caller.

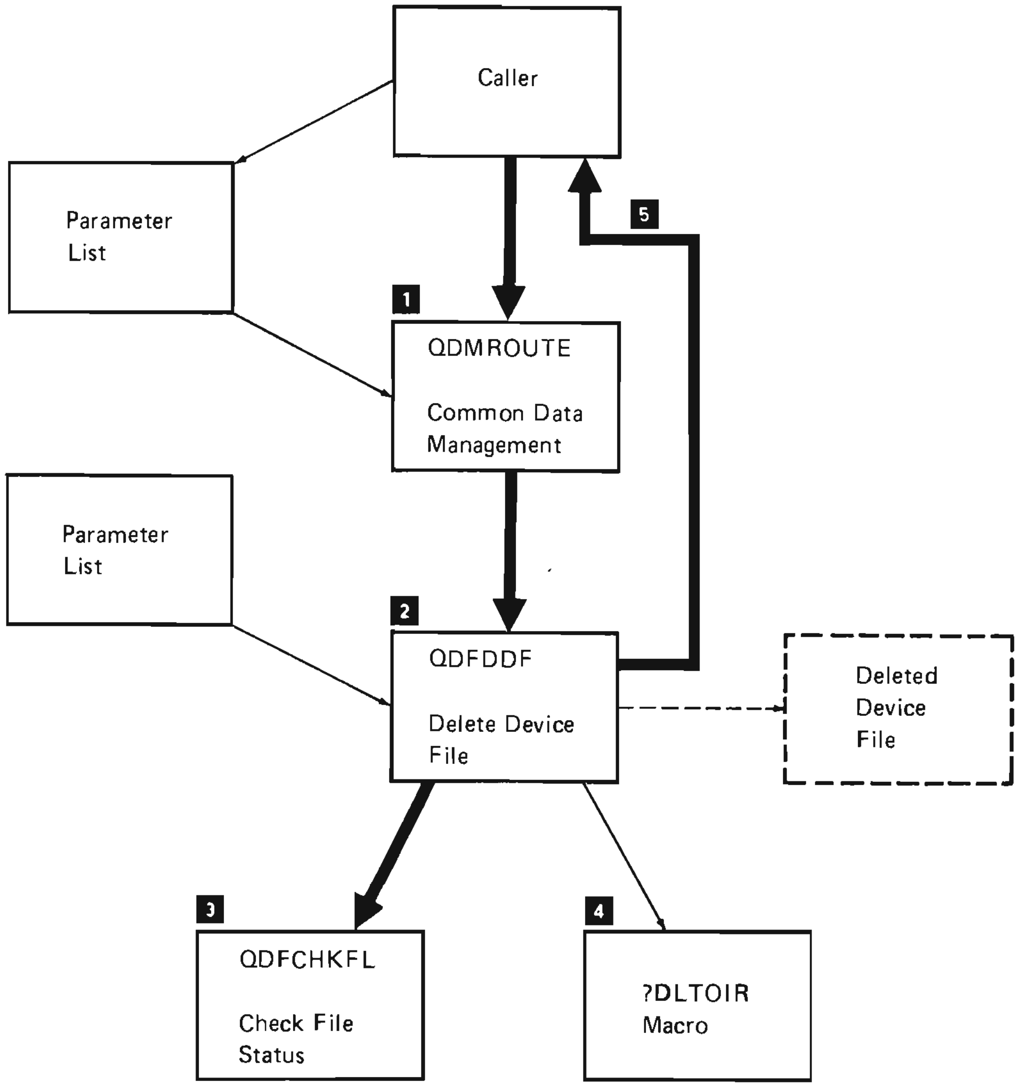


Figure DF-4. Delete Device File Definition Overview

Device File Definition Extract Operation

Figure DF-5 and the following text describe a device file definition extract information operation.

1 The file reference function component, high-level languages, and other functions call the QDMROUTE module of common data management via the ?EXTFILED macro. A parameter list is passed that contains a pointer to a structure that contains either a null pointer or a user-defined pointer. When control is returned to the caller, this pointer will point to a space that contains the extracted information. The input structure must identify the type of extract to be processed:

- File attributes
- Name list of all record formats in the file
- Specific record format description
- Specific field description in a record format

The qualified name of the file from which the information is to be extracted is also in the parameter list.

2 If information is to be extracted from a device file, QDMROUTE calls QDFEDF to extract the information. The parameter list that was passed to QDMROUTE is passed to QDFEDF as well as a resolved system pointer to the file that contains the information to be extracted.

3 Librarian is invoked to extract the file text from the OIR.

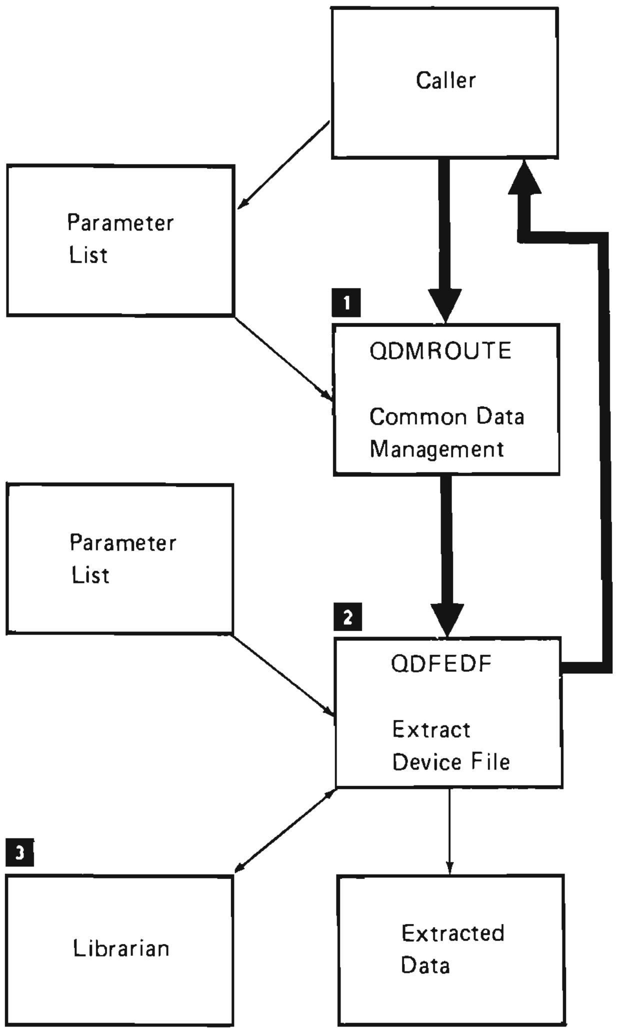


Figure DF-5. Device File Definition Extract Operation Overview

This page is intentionally left blank.

Convert Device File Definition

Figure DF-6 and the following text describe the device file definition convert operation.

1 QDFCVALL is called by the ?CNVFILES macro. A parameter list is passed that contains pointers to the files to be converted. This generally occurs at installation time.

2 QDFCNVPP is invoked to maintain the current release/modification level for the device files on the system.

3 QDFCNVF is called by the ?CNVDF macro. QDFCNVF is invoked for each device file that is converted to the current release/modification level. This is generally performed by save/restore and reclaim/damage notification.

4 QDFMATR1 or QDFMATR2 is invoked to materialize device files not at the current release/modification level into a form that can be used in the remainder of the process. QDFMATR1 is invoked for display and printer files. QDFMATR2 is invoked for communications and BSC files.

5 QDFCNVP calls QDFBLDDF to build the device file.

6 Common data management is invoked to initialize the prototype ODP (open data path) contained in every file.

7 QDFBLDDF calls QDFLVLGN to generate the level check values for the device file.

8 QDFBLDDF calls QDFRCDPR to process the record formats.

9 QDFRCDPR calls QDFKWDPR to process any keywords specified through the data description specifications.

10 QDFRCDPR calls QDFCRTWU to create the where-used section used by extract.

11 QDFLVLGN calls QDFINIT to initialize keyword tables.

12 QDFBLDDF calls QDFCDFPR to create and process the record formats, fields, and keywords for the externally described printer and display files. QDFCDFPR also builds the where-used section used by the extract function.

13 QDFBLDF calls QDFDFTPR to create non-field level device files.

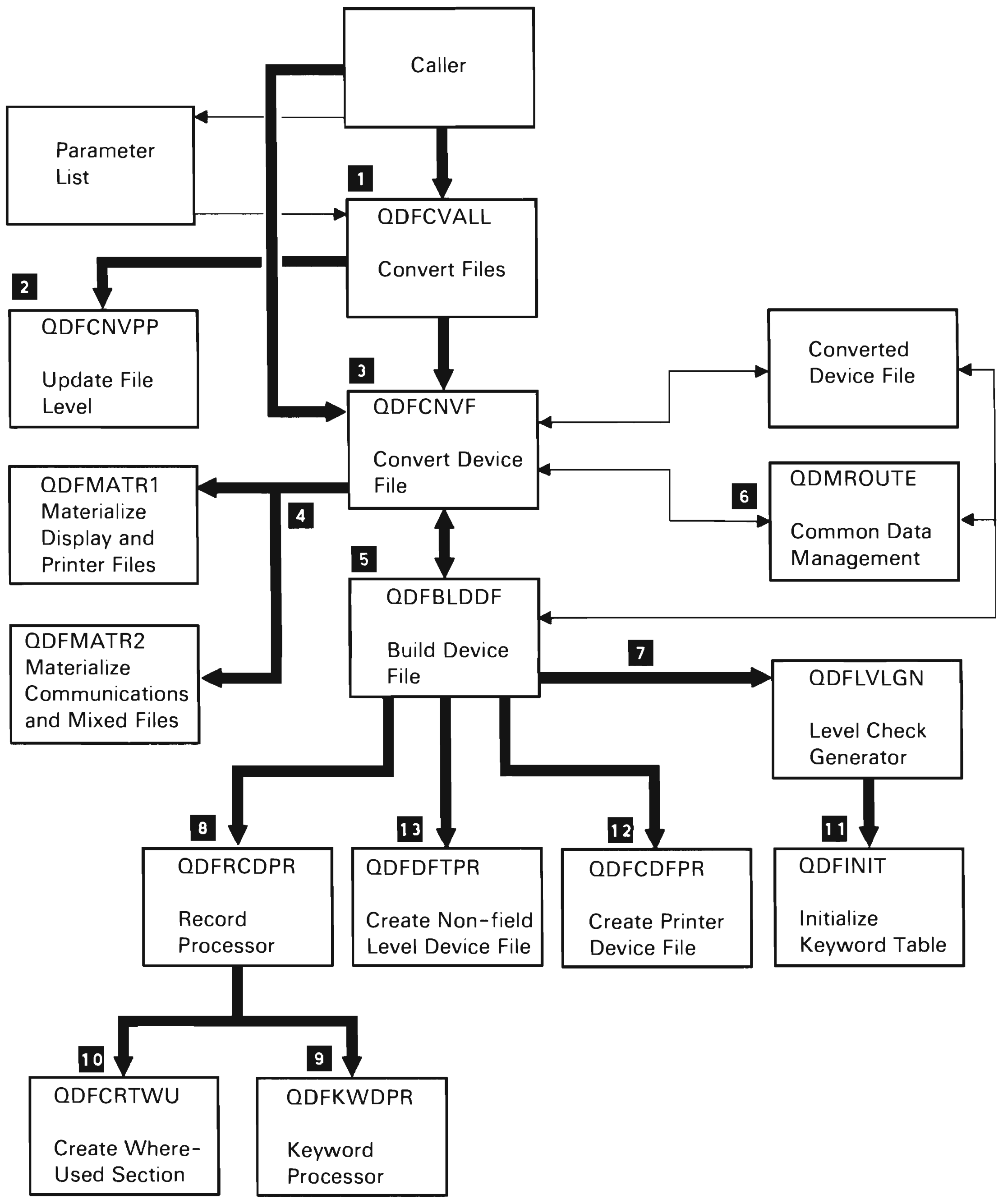

Caller
Parameter List
1
QDFCVALL Convert Files
2
QDFCNVPP Update File Level
3
QDFCNVF Convert Device File
Converted Device File
QDMROUTE Common Data Management
6
QDFMATR1 Materialize Display and Printer Files
4
5
QDFBLDDF Build Device File
QDFMATR2 Materialize Communications and Mixed Files
7
QDFLVLGN Level Check Generator
8
QDFRCDPR Record Processor
13
QDFDFTPR Create Non-field Level Device File
12
QDFCDFPR Create Printer Device File
11
QDFINIT Initialize Keyword Table
10
QDFCRTWU Create Where-Used Section
9
QDFKWDPR Keyword Processor
PAAB044-0

Figure DF-6. Convert Device File Definition Overview

Device File Definition Subset Operations

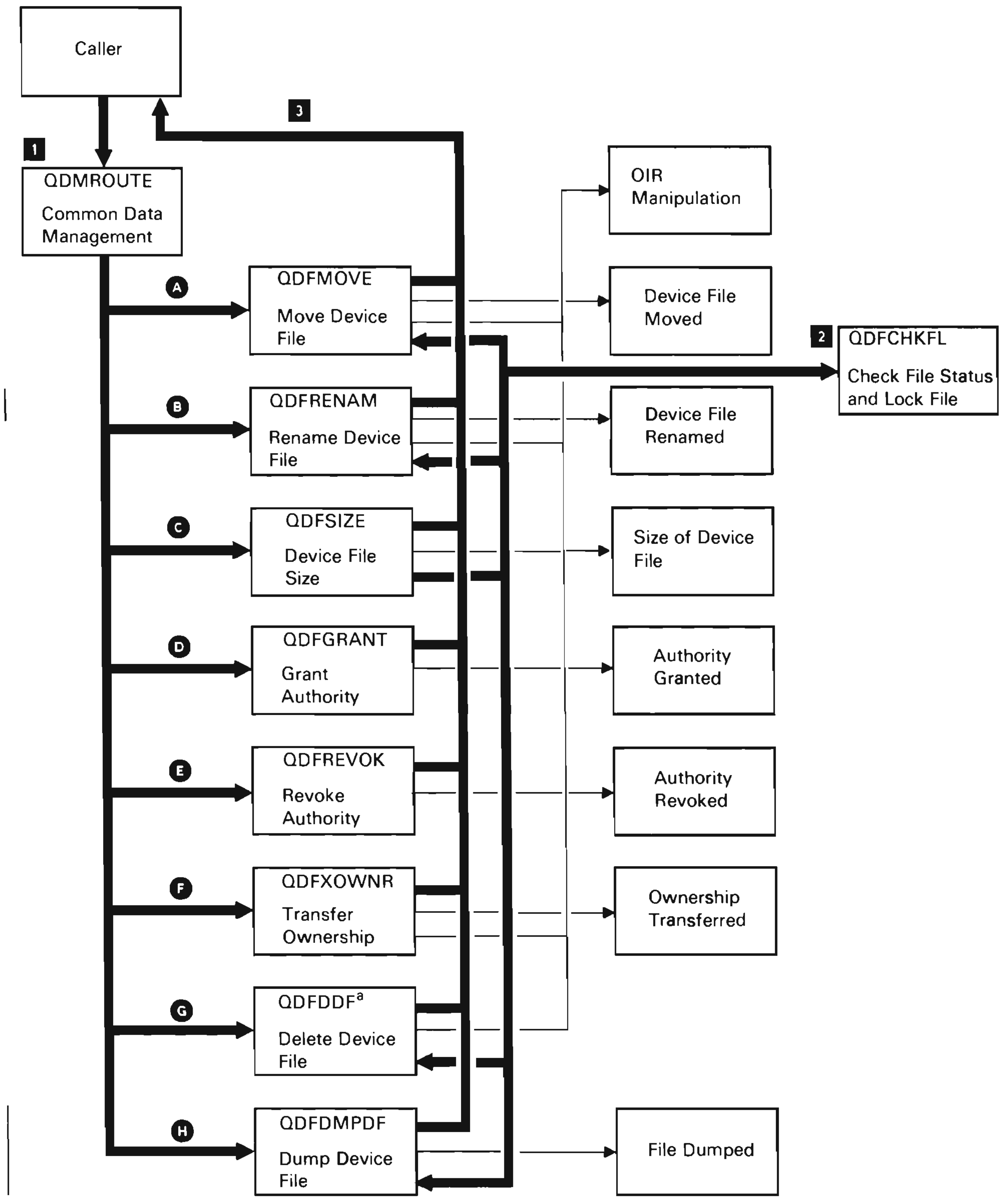
Figure DF-7 and the following text describe the functions of the subset of operations for device file definition.

1 The security, librarian, or service component and other functions call the QDMROUTE module of common data management. If the operation is to be performed on a device file, QDMROUTE transfers control to:

- A QDFMOVE if a device file is to be moved
- B QDFRENAM if a device file is to be renamed
- C QDFSIZE if the size of a device file is to be determined
- D QDFGRANT if authority is to be granted to a device file
- E QDFREVOK if authority is to be revoked to a device file
- F QDFXOWNR if ownership of a device file is to be transferred
- G QDFDDF if a device file is to be deleted
- H QDFDMPDF if a device file is to be dumped

2 QDFMOVE, QDFRENAM, QDFSIZE, and QDFDDF, calls QDFCHKFL to status check and obtain/release locks on objects involved in the requested operation.

3 Control is returned to the caller.



[a] See Figure DF-4.

PAAB038-0

Figure DF-7. Device File Definition Subset Operations Overview

Duplicate Device File Operation

Figure DF-8 and the following text describe a duplicate device file operation.

1 The librarian calls QDFDUPDF to duplicate an online save file. A parameter list is passed that contains a pointer to a structure containing a pointer to the file to be duplicated and the new file and library names.

2 The ?EXTFILED macro is invoked to extract the file level attributes of the online save file.

3 The ?CRTDEVF macro is invoked to create a new online save file.

4 QDFDUPDF calls QDFCHKFL to return a lock on the newly created file.

5 Control is returned to the caller.

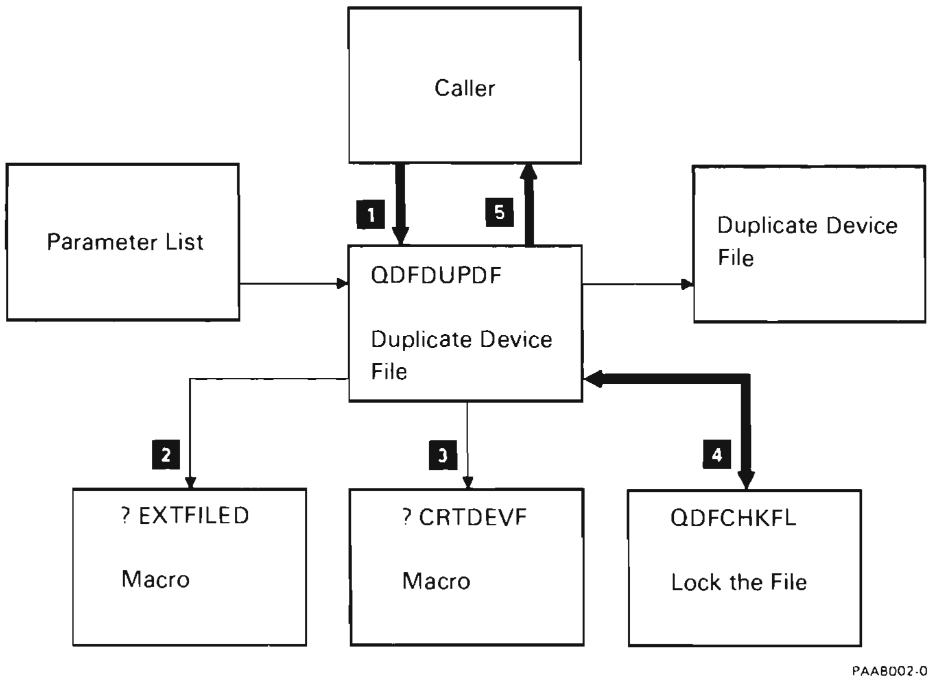


Figure DF-8. Duplicate Device FIle Operation

Save/Restore of an Online Save File

Figure DF-9 and the following text describe the save/restore handling of an online save file.

1 Save calls QDFSAVDF to save the file level attributes of an online save file. A parameter list is passed that contains a pointer to a structure containing the file and library names and a pointer to the file.

2 The ?EXTFILED macro is invoked to extract the file level attributes of the online save file.

3 Control is returned to the caller. The parameter list structure is updated to contain a pointer to the extract space.

4 Restore calls QDFRSTDF to create a new or to modify an existing online save file. A parameter list is passed containing a pointer to the file (if it exists), a pointer to the new owner's user profile, and a pointer to the extract space.

5 The ?CRTDEVF macro is invoked to create a new online save file.

6 QDFRSTDF calls QDFCHKFL to lock the newly created file. This lock will be returned to the caller.

7 QDFRSTDF calls QDFREVOK to revoke all authority from the process user profile and private authority of the original owner.

8 QDFRSTDF calls QDFXOWNER to transfer ownership to the requested owner.

9 QDFRSTDF calls QDFGRANT to grant public authority, if any, to the file.

10 Control is returned to the caller. The parameter list structure is updated to contain a pointer to the newly created file.

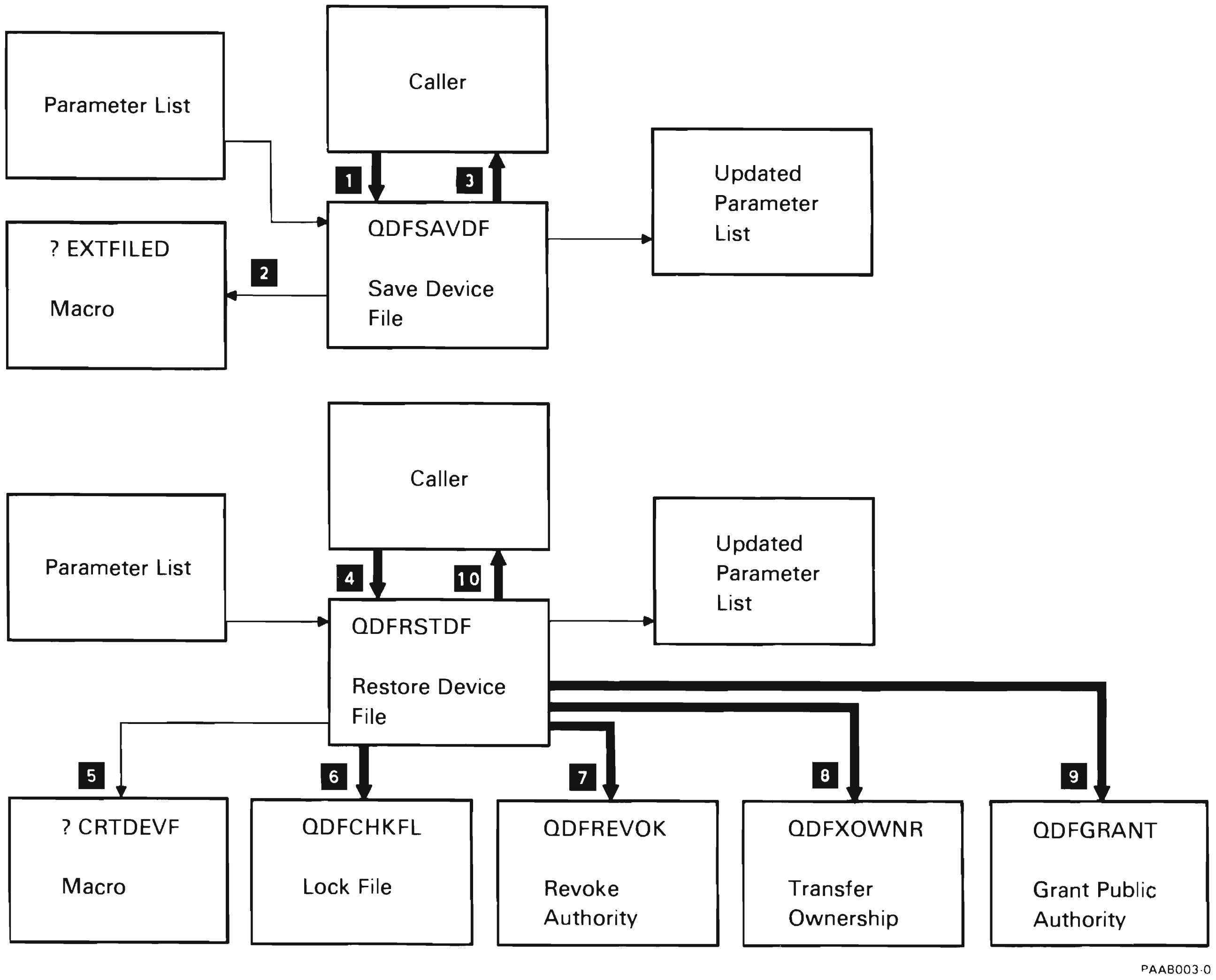


Figure DF-9. Save/Restor of an Online Save File

Diskette Function Manager

INTRODUCTION

The diskette function manager component of the CPF (control program facility) provides the support for the diskette device on System/38.

The diskette is a magnetic diskette storage device that is supported as a system I/O device, data interchange device, and save/restore device. It contains two magazines, each of which can contain ten diskettes, and three slots for individual diskettes.

The following diskette functions are supported by the diskette function manager:

- Initialize volume
- Display volume
- Duplicate a volume
- Rename volume
- Clear volume
- Delete diskette file
- Check volume for a specific volume and file label
- Open diskette file for processing
- Close diskette file to processing
- Read data from a diskette file
- Write data to a diskette file
- End-of-volume processing

GENERAL OVERVIEW

Diskette Function Manager Modules

The diskette function manager component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QDKDSPY–Display Volume (DSPDKT)[1]: This module displays the diskette volume and file labels.

-->QDKDUP–Duplicate Diskette (DUPDKT)[1]: This module duplicates diskette volume(s).

-->QDKOPEN–Diskette Open: This module opens a file VTOC or opens a file for input or output processing.

-->QDKINZFY–This module is called by QDKOPEN to initialize a diskette during volume open. It will check for active files and initialize the diskette to a usable format and volume identifier.

-->QDKGET–Diskette Get: This module retrieves one or more records from a BASIC or H exchange file for the user.

> QDKEOV–Diskette End-of-Volume Processing: This module closes the current volume when an end-of-volume condition is detected and then opens the next volume for a multivolume operation.

-->QDKGETI–Diskette Get: This module retrieves one or more records from an I exchange file for the user.

> QDKEOV–Diskette End-of-Volume Processing: This module closes the current volume when an end-of-volume condition is detected and then opens the next volume for a multivolume operation.

[1]This module is a CPP (command processing program).

-->QDKPUT–Diskette Put: This module writes one or more records of user data to a BASIC or H exchange file on the diskette.

QDKEOV–Diskette End-of-Volume Processing: This module closes the current volume when an end-of-volume condition is detected and then opens the next volume for a multivolume operation.

-->QDKPUTI–Diskette Put: This module writes one or more records of user data to an I exchange file on the diskette.

QDKEOV–Diskette End-of-Volume Processing: This module closes the current volume when an end-of-volume condition is detected and then opens the next volume for a multivolume operation.

-->QDKCLOSE–Diskette Close: This module closes a file to input or output processing.

QDKEOV–Diskette End-of-Volume Processing: This module closes the current volume when an end-of-volume condition is detected and then opens the next volume for a multivolume operation.

-->QDKFEOD–Forced End-of-Data: This module signals the end-of-file for an input file. It has no function for an output file.

QDKEOV–Diskette End-of-Volume Processing: This module closes the current volume when an end-of-volume condition is detected and then opens the next volume for a multivolume operation.

-->QDKUTIL–Diskette Utilities[1]: This module provides the support for the following commands:

- Initialize a Diskette (INZDKT)
- Rename a Diskette (RNMDKT)
- Clear a Diskette (CLRDKT)
- Delete a Diskette File Label (DLTDKTLBL)

QDKERROR–Diskette Errors: This module signals exceptions and sends messages for conditions detected by the diskette function manager.

QDKLUDIN–Diskette LUD Initialization: This module resets the user-defined portion of the LUD-associated space.

-->QDKCHECK–Check Diskette (CHKDKT)[1]: This module is used to check for the first occurrence of a specific diskette volume, file label, or file on a specified volume within a given diskette.

QDKCHEXT–Diskette Invocation Exit Program: This module is invoked to close a diskette file if QDKCHECK is bypassed because of normal exception handling, or process termination.

Diskette Operation

Figure DK-1 and the following text describe a diskette operation.

1 A high-level language program, the spooling, copy, or save/restore component, through the QDMCOPEN module of common data management, calls QDKOPEN to open a diskette file for input or output processing.

A An argument list is passed that contains a pointer to the UFCB (user file control block).

The diskette to be used is selected by the value in the LOC parameter specified by the caller of common data management. The volume label identifier field is verified if the caller specified a volume ID.

If the file is being opened for input:

- The diskette file labels are searched for a match to the file name specified by the caller. If a creation date is specified, the labels are searched for a match of both the file name and creation date specified by the caller.

B A message is sent to the system operator console if the file cannot be found. The operator can insert another diskette and retry the operation or the job can be canceled.

[1]This module is a CPP (command processing program)

- The file record length, as specified in the file HDR1 label, is compared to the value specified by the caller. If the record length specified by the caller is longer than the record length of the file, a diagnostic is signaled to the caller. Processing of the file continues, but each record read will be padded with blanks on the end. If the record length specified by the caller is shorter than the record length of the file, a diagnostic is not signaled to the caller. Processing of the file is continued, but each record read will be truncated.

Note: For files in basic exchange and H-exchange, the file record length is determined by the block length field in the file label. For files in I-exchange and E-exchange, the file record length is determined by the record length field in the file label.

C Request I/Os are issued to fill both buffers.

- Two buffers are used by the diskette function manager. Each buffer holds one diskette track of data. For I-exchange files, additional space preceding or following the buffer may be used to hold spanning records.

If the file is being opened for output:

- All expired files are deleted from the VTOC (volume table of contents) of the selected diskette. A file is considered to be expired if the file expiration date (in the file HDR1 label) is less than or equal to the system date.

- The file labels on the diskette are searched to verify that the name of the file to be written (specified by the caller in the LABEL parameter) is unique.

A If a file name having the same name as the file to be written is found on the diskette, a message is sent to the system operator queue. The operator can insert another diskette and retry the operation or the job can be canceled.

- Space for the new file is allocated immediately following the last unexpired file on the diskette. (The last unexpired file is the one having the highest diskette address.)

B If space is not available for the new file, a message is sent to the operator. The operator can insert another diskette and retry the operation or the job can be canceled.

- A label for the new file is built for the diskette VTOC but is not written to the diskette until the file is closed or an EOV (end-of-volume) occurs. Space allocated for the file is noted in the label BOE (beginning of extent) and EOE (end of extent) fields as occupying all of the space from the last unexpired file to the end of the diskette. When the file is closed, the EOE field is updated to show the actual end of the file.

- For files in basic exchange or H-exchange, the diskette sector size is compared to the record length specified by the caller. If that record length is longer than the diskette sector size, a diagnostic message is signaled to the caller. Processing of the file continues, but each record written will be truncated. If the record length is shorter, a diagnostic message is not signaled. Processing is continued, but each record written is padded with zeros. For files in I-exchange, the records span sectors up to a record length of 4 096. Records are written contiguously in a sector, regardless of the size of the record or the size of the sector.

2 After the file has been opened, information is written to the file by calling QDKPUT or QDKPUTI.

A An argument list is passed that contains pointers to the UFCB, an option list, and control information.

B The option list and control information are ignored to the extent that a put wait operation can be requested; other requests result in an error message being sent to the caller.

C Request I/Os are issued to the diskette I/O manager when the user has sent the diskette function manager enough records to write a diskette track of data.

Note: The save/restore component does not use this interface. Save/restore issues special request I/Os to put data to a diskette. See *Save/Restore.*

3 After the file has been opened, information can be retrieved from the file by calling QDKGET or QDKGETI.

A An argument list is passed that contains pointers to the UFCB, an option list, and control information.

B The option list and control information are ignored to the extent that a get next wait operation can be requested; other requests result in an error message being sent to the caller.

C Request I/Os are issued when the caller has emptied a buffer. There are two buffers; each buffer contains a diskette track of data.

Note: The save/restore component does not use this interface. Save/restore issues special request I/Os to retrieve data from a diskette. See *Save/Restore*.

4 After a file has been processed, it is closed by calling QDKCLOSE through QDMCLOSE.

A An argument list is passed that contains pointers to the ODP (open data path), an index to the device being closed, and the type of close to perform (permanent or temporary).

If the file being closed had been opened as an output file:

- The data remaining in the buffers is written to the file.
- The file HDR1 label in the VTOC buffer is updated to reflect the true end of the file, and is written to the diskette. (This frees up space past the end of the file being closed so that the space remaining can be allocated to subsequent output files.)

If the file being closed had been opened as an input file:

- The diskette I/O manager is instructed by a Reset command to stop processing any current or pending request I/Os.
- The VTOC is not updated because it is not necessary to do so for an input file.

If a permanent close is requested, all objects created by the diskette function manager are destroyed.

5 When a forced end of data is requested, QDKFEOD is called and the following occurs:

A An argument list is passed that points to the UFCB.

B If the file is opened for input, an end-of-file exception is signaled to the user. (For multivolume files, the end-of-file exception is signaled after the last volume of the file has been located.)

If the file is opened for output, the operation is ignored.

6 End-of-volume switching occurs automatically within the diskette function manager when it is processing a multivolume file.

If a file is open for output, QDKPUT or QDKPUTI calls QDKEOV to perform the volume switch. If a file is open for input, QDKGET or QDKGETI calls QDKEOV to perform the volume switch. If a file is being closed, QDKCLOSE calls QDKEOV to perform the volume switch when there is not enough space to write the data remaining in the buffers.

The save/restore component calls QDKEOV when it detects an end-of-volume condition during input or output save/restore operations or if a media error occurs during output.

If a file is open for output, QDKEOV:

- Calls QDKCLOSE to update the file label in the VTOC of the current volume to reflect that the file is being continued on another diskette volume.

- Calls QDKOPEN to increment the diskette magazine and load the next diskette. QDKOPEN performs the checks as described for output files and builds the file HDR1 label in the VTOC. The HDR1 label also contains a volume sequence number that will be one unit higher than the number written in the previous volume of the file.

- The portion of the buffer that could not be written to the previous volume is now written to the new volume.

If the file is open for input, QDKEOV:

- Calls QDKOPEN to increment the diskette magazine and load the next diskette. The VTOC of the new diskette is searched for the HDR1 label of the continued file.

- The volume sequence number in the HDR1 label is checked to verify that the next volume of the file is in proper sequence.

B A message is sent to the operator console if the test fails. The operator can insert another diskette and retry the operation, ignore the condition and process the diskette with incorrect volume sequence number, or the job can be canceled.

- QDKOPEN causes a seek to the start of the file.

- The record obtained from the new diskette volume is returned to the caller of QDKGET.

End-of-volume processing causes a notify message to be sent to the caller of get or put. The message, which can be ignored, informs the user that a volume switch occurred and processing continued on the next diskette.

End-of-file processing causes a status message to be sent to the caller of get. If this message is ignored, an escape message is sent to the caller of get.

B The message handler is called to send messages to the operator console and to signal exceptions to the user. Information is also written to the job log that pertains to a particular error or exception.

C Request I/Os communicate to the diskette I/O manager the desired action. The diskette I/O manager indicates its success or failure in performing the request by returning a message in the machine interface response queue.

QDKERROR signals all operator messages and program exceptions. It is also called to analyze I/O errors to determine what recovery action is to be performed.

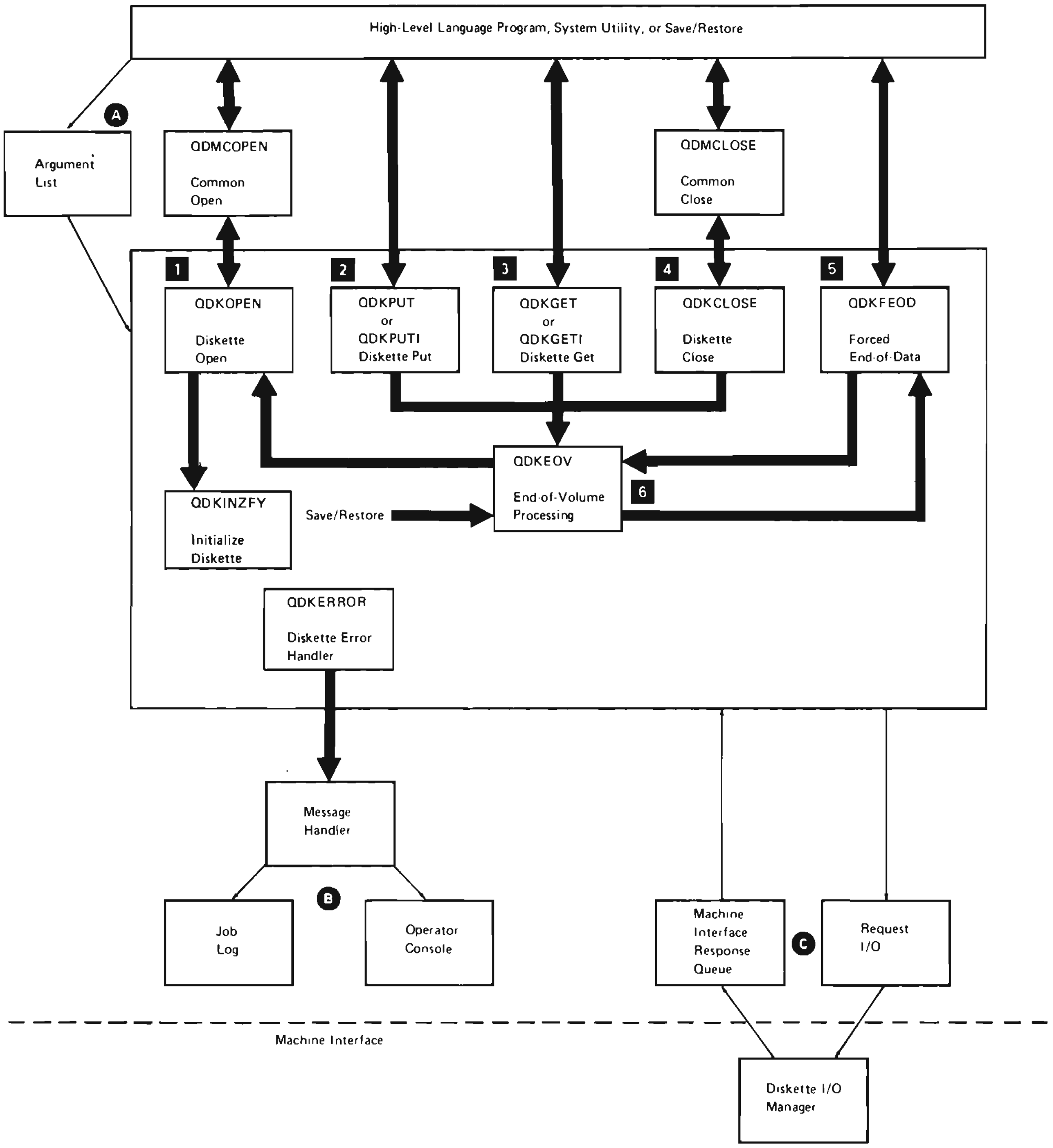

High-Level Language Program, System Utility, or Save/Restore
A
Argument List
QDMCOPEN Common Open
QDMCLOSE Common Close
1
QDKOPEN Diskette Open
2
QDKPUT or QDKPUTI Diskette Put
3
QDKGET or QDKGETI Diskette Get
4
QDKCLOSE Diskette Close
5
QDKFEOD Forced End-of-Data
QDKEOV End-of-Volume Processing
6
QDKINZFY Initialize Diskette
Save/Restore
QDKERROR Diskette Error Handler
Message Handler
B
Job Log
Operator Console
Machine Interface Response Queue
C
Request I/O
Machine Interface
Diskette I/O Manager

Figure DK-1. Diskette Operation Overview

Common Data Management

INTRODUCTION

The common data management component of the CPF (control program facility) is used to help manage data that is to be processed by programs. Common data management can be used to format data into records, organize data records into files, and transfer the records of a file between a program and the file.

The functions that make up the common data management component are:

- The common parts of open and close that are common across device support and the data base or across different devices
- The common data management macros–?OPEN, ?CLOSE, ?GET, ?PUT, ?UFCB and so forth
- ODPs (open data paths)
- Overrides
- Acquire program devices
- Release program devices
- Locking
- Unlocking
- Accept input
- Routing common functions
- Pass device

Common data management is first invoked during device file and member creation to construct the inactive ODP. The inactive ODP is created as a part of the complete device file or as a part of the interactive data base cursor/member. Figures DM-1 and DM-2 show the ODP structure after file or member creation.

GENERAL OVERVIEW

Common Data Management Modules

The common data management component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QDMACQDV–Acquire Program Device: This module activates a program device in an open device file ODP and creates a lock acknowledgement event handler.

-->QDMNODEV–Signal Program Device Not Found: This module signals an escape exception if the program device specified on an I/O macro cannot be found in the device name list in the ODP.

-->QDMACCIN–Accept Input: This module accepts input from the first invited program device in an open device file that has data available.

-->QDMACQDP–Acquire Display: This module adds and activates a display device to an open display file ODP.

-->QDMACKEH–Lock Acknowledgement Event Handler Program: This module handles the lock acknowledgement event for QDMACQDV and QDMLOCK, cancels the event monitor that invoked QDMACKEH, and unlocks the associated device.

-->QDMCOPEN–Data Management Common Open: This module establishes an ODP between the calling program and a data base file, physical device, or logical (spooled) device.

> QDMGETOV–Get Overrides: This module is used to find any overrides that might exist for the file being opened.

QDMEHDES–Resolve Processing: This module gets addressability to devices for device files after the device file has been saved and restored or after the device has been destroyed and re-created.

QDMMINIT–Merge Initialization: This module initializes the tables necessary to merge parameters from the UFCB (user file control block) or an override when a file is opened.

QDMSIGNL–Send Escape Message: This module sends an escape message to the caller of QDMCOPEN, if an error occurs while opening a file.

-->QDMOVERD–Create Override Control Block (OVRBSCF, OVRCMNF, OVRCRDF, OVRDBF, OVRDKTF, OVRDSPF, OVRMSGF, OVRMXDF, OVRSAVF, OVRPRTF, OVRTAPF)[1]: This module creates an OCB (override control block) with the parameter values specified on the command.

-->QDMDSPOV–Display Override (DSPOVR)[1]: This module displays the override information and the invocation level in which the command was submitted. It displays either the override parameters for a single OCB or a list of all the override file names. This module is also invoked when option 11 is taken on the DSPJOB display.

-->QDMDELOV–Delete Override (DLTOVR)[1]: This module deletes either a single OCB or all the OCBs specified in the same invocation level or in a CL program.

-->QDMLOCK–Lock Data Management and System Objects: This module allocates to a process any system object or all of the required objects to process a data base file member or a device file, and creates a lock acknowledgement event handler.

-->QDMTCLSE–Data Management Termination Close: This module closes all open files at process termination and resets the DMCQ (data management communications queue) and MIRQ (machine interface request queue).

-->QDMUNLCK–Unlock Data Management and System Objects: This module deallocates any system object or all of the required objects needed to process a data base file member or a device file.

-->QDMCRODP–Create ODP: This module creates an inactive ODP.

-->QDMROUTE–Extract Override and Route: This module provides a single common data management interface used for creating, modifying, deleting, renaming, moving, transferring ownership of, granting authority to, and extracting both data base and device files.

-->QDMIFERR–Interface Error: This module gets control and signals an exception when an operation that is not valid is attempted to a file.

-->QDMPASS–Pass Device: This module transfers the allocation of a device from one process to another process.

-->QDMBKOUT–Backout: This module closes files or releases program devices after an error is detected and the escape message was not monitored.

-->QDMRLSDV–Release Program/Display Device: This module releases a program or display device from an open device file ODP.

-->QDMCLOSE–Data Management Common Close: This module closes the files specified by the ?CLOSE macro.

-->QDMERRHP–Error Handler Program: This module supplies message data for all escape and notify messages and sets the ODP to the error state for all escape messages.

-->QDMRCLSE–Reclaim Close: This module closes all files open in a process that were opened at an invocation number greater than the invocation number passed to the module.

-->QDMDSPOF–Display Open Files: This module displays information about the files that are currently open in the specified routing step. This module is invoked when option 10 is used on the DSPJOB display.

[1]This module is a CPP (command processing program).

Open

Open is the process of making an inactive ODP active and preparing it so that it can perform I/O operations. When a file is opened, common data management performs the following functions:

- Determines if overrides are to be applied to the file
- Gets addressability to the file to be opened
- Creates a copy of the inactive ODP in the process access group
- Applies the parameters from the UFCB and the override, if they exist
- Allocates a device or data spaces to the process for the file
- Performs level checking
- Sets the event monitors for those files that have been specified NOWAIT(*YES) or were created with the INVITE keyword
- Updates the UFCB and open feedback area
- Invokes the device or data base open routine

Close

Close is the process of deactivating an active ODP and destroying the temporary objects. There are three interfaces to the close routines. One is invoked in a program by the ?CLOSE macro (QDMCLOSE). The second one is invoked either by the user via the Reclaim Resources (RCLRSC) command or by the system via the ?RCLFILE macro (QDMRCLSE). The third one is invoked on behalf of the user by the system (QDMTCLSE). Close performs the following functions:

- Invokes the device or data base close routines
- Signals switched lines closed event
- Deallocates the objects allocated by open
- Cancels the event monitors for those files that have been specified NOWAIT (*YES) or were created with the INVITE keyword
- Updates the UFCB to show the file is closed if the interface is through the ?CLOSE macro or the RCLRSC command

Device File Definition

The Create Device File commands (CRTBSCF, CRTSAVF, CRTCMNF, CRTCRDF, CRTDKT, CRTDSPF, CRTMXDF, CRTTAPF) cause a device file to be created. The first section of a device file space object is the inactive ODP. Figure DM-1 shows a device file space object.

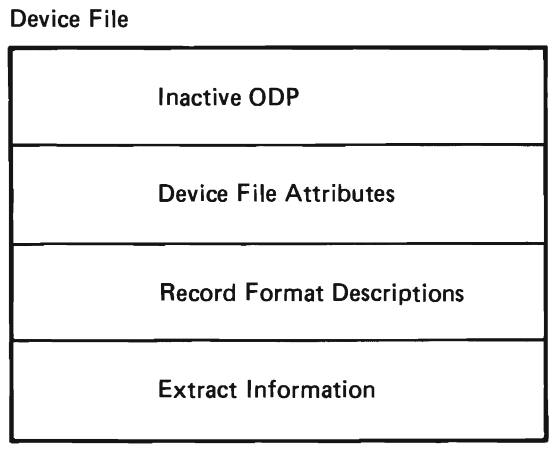


Figure DM-1. Device File Space Object (Before Open)

Data Base File Definition

The Create Physical File (CRTPF), Create Logical File (CRTLF), Add Physical File Member (ADDPFM) and Add Logical File Member (ADDLFM) commands create the FCB (file control block) and the prototype cursor/ODP/member. Figure DM-2 shows the structure of a data base file or member before open.

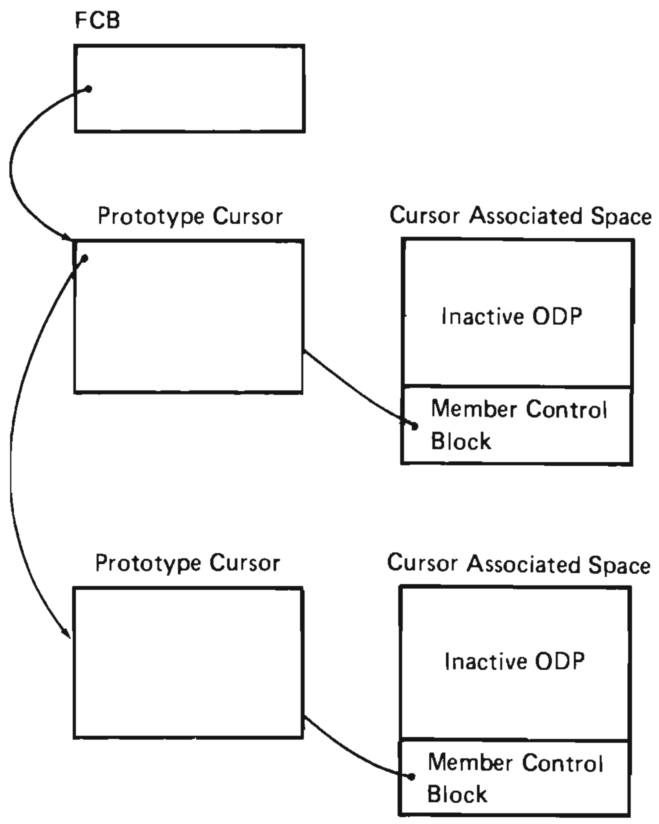


Figure DM-2. Data Base File/Member Structure (Before Open)

Structure of the Common Data Management Objects after Opening a Device File

Figure DM-3 and the following text describe the structure of the common data management objects after a device file is opened.

The UFCB for the file can reside in a separate space that is either permanent or temporary, or it can be declared in the program and reside in static (PSSA) or automatic (PASA) storage.

Includes for the ODP for all file types are provided by common data management. Data description specifications provide includes for device file attributes section of the ODP (existing includes for device file attributes).

The space objects that contain the source/sink request and the source/sink data should specify a transfer size based on the number of source/sink requests and source/sink data in the object.

The MIRQ is an extendable queue that is outside the process access group because it is referenced by both CPF tasks and the machine. File-dependent opens get addressability to the queue for the source/sink requests from the WCB (work control block).

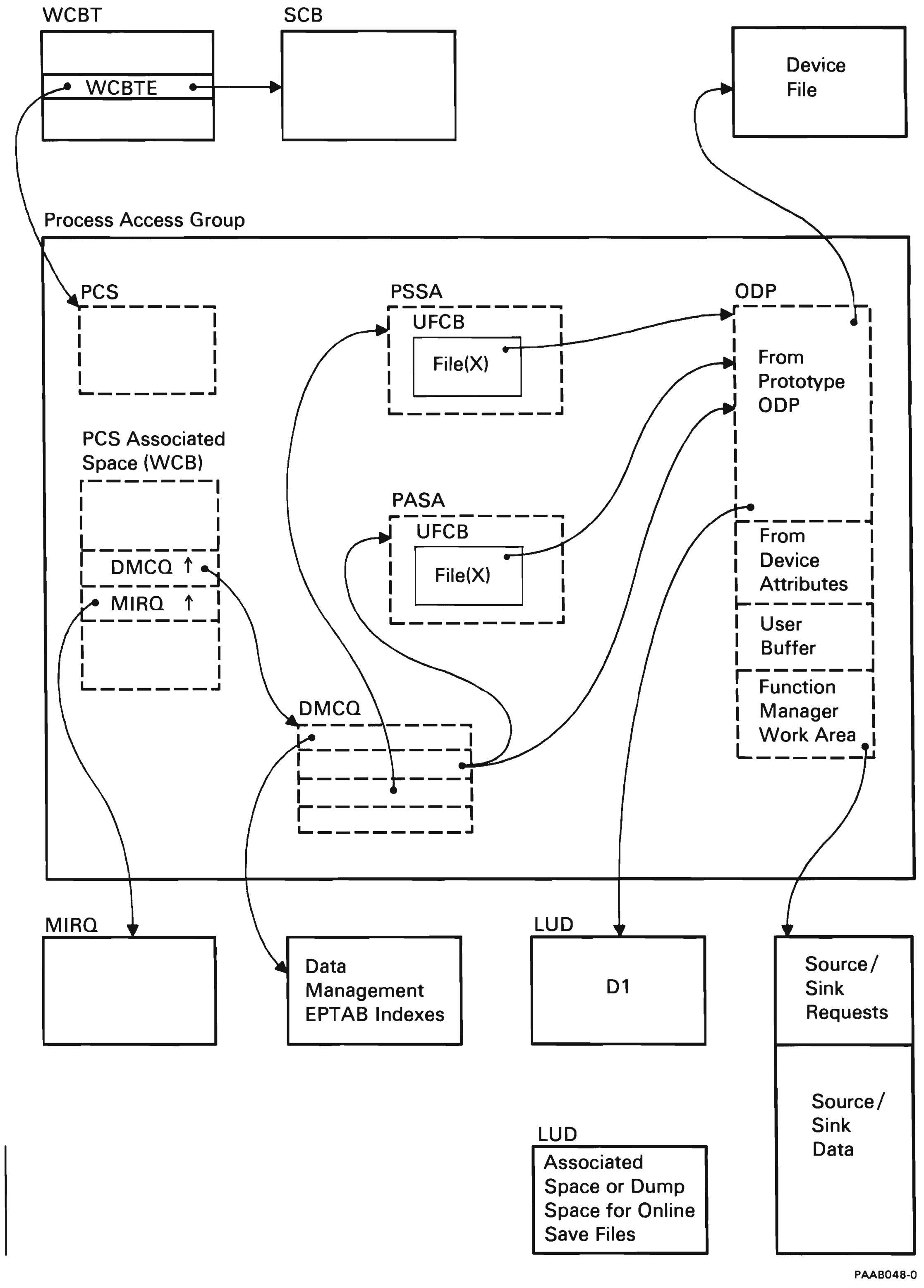


Figure DM-3. Structure after Opening a Device File)

Structure of the Common Data Management Objects after Opening a Multi-Device File

Figure DM-4 and the following text describe the structure of the common data management objects after a multi-device file is opened.

The number of devices that can be attached to a device file is specified on a create command and can be changed by a Change Device File command. A change in the number of devices causes a re-creation of the ODP. This number is used at open time and when a device is added to the file.

The device open routines calculate the space for the function manager work areas and must get one user buffer for the file, one function manager work input area, and optionally a user output buffer for each device connected to the file.

The device-dependent open routines will be called for each device specified in the open parameters. Devices are activated one at a time to an open device file using acquire program device. Acquire program device will attach the specified program device if the file is open.

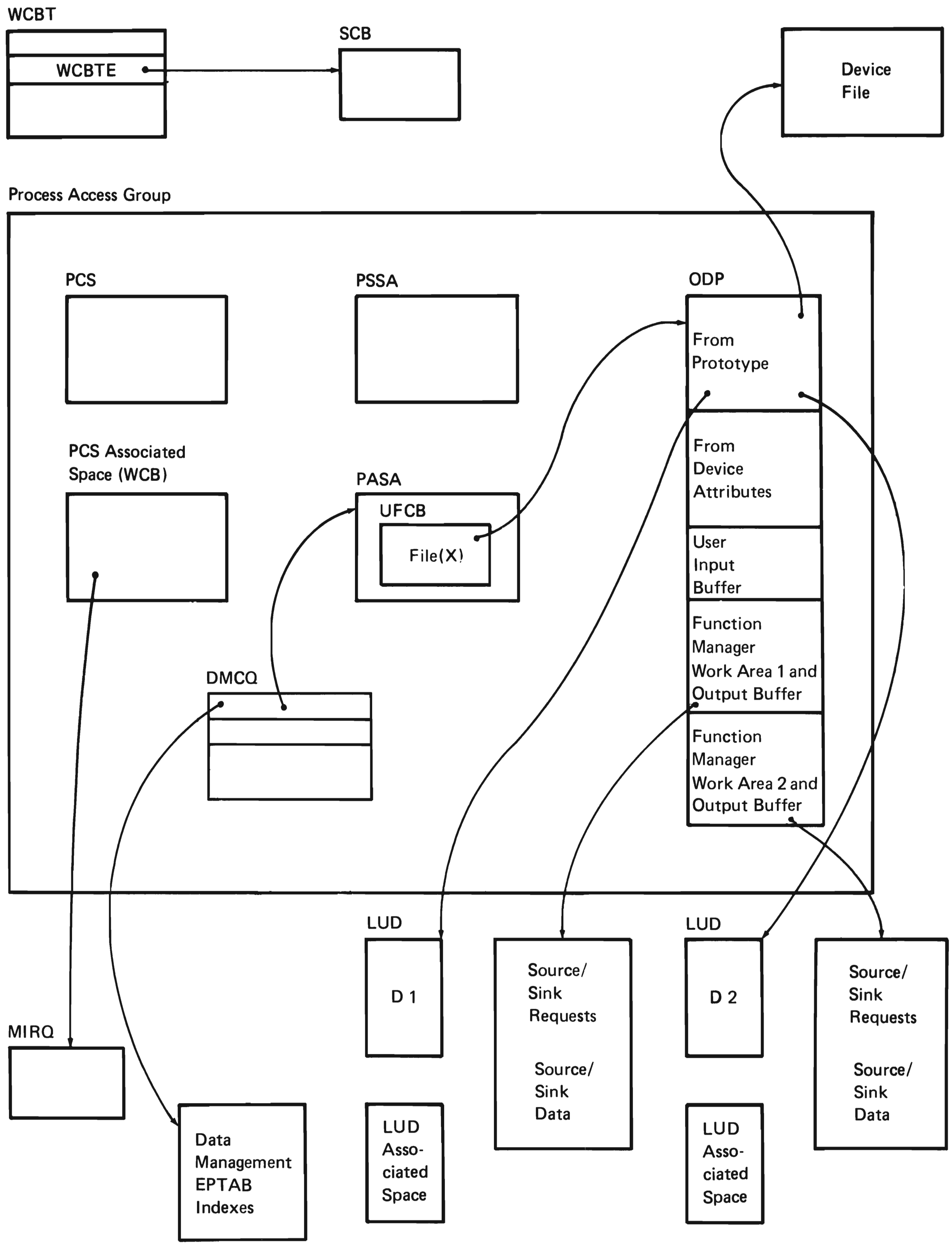


Figure DM-4. Structure after Opening a Multi-Device File)

Common Data Management Macros

The common data management macros are used to invoke the related common data management functions. The ?OPEN, ?CLOSE, and ?UFCB macros interface with common data management functions directly. The ?GET, ?PUT, ?PUTGET, and similar macros interface with the function managers for the file. These macros are common data management macros because they interface to more than one type of file.

Override

The override commands (OVRBSCF, OVRCMNF, OVRCRDF, OVRDBF, OVRDKTF, OVRDSPF, OVRMSGF, OVRMXDF, OVRPRTF, OVRSAVF, OVRTAPF, DSPOVR, and DLTOVR) provide a full range of functions to let the user control overrides active in the invocation or CL program.

Acquire Program Device

Acquire program device adds a program device to an opened file and opens the device.

Release Program Device

Release program device will disconnect a device from an opened ODP. To release a device, the device is closed. All devices of a file can be released, leaving the ODP opened but set to a condition where I/O operations are not allowed. Releasing a device can also deallocate the device from the process.

Locking/Unlocking

Common data management locking provides an interface for a user to lock/unlock the complex objects of common data management as an atomic operation. (An atomic operation is an operation that, once started, must continue to completion without interruption.) Also, if a device belongs to another process and that device can be obtained, the lock function will obtain the device.

Accept Input

The common data management accept input waits on data. The data arrives from a request by a get nowait, a put-get nowait, or a put with invite. The nowait functions overlap program execution with the user I/O requests.

Pass Device

The pass device is used to transfer a device from one process to another process without losing allocation to a third process.

Routing Common Function

The common data management routing function allows all CPF commands that have a generic function for data management objects to be routed to the correct module to perform the function.

Install Object

Common data management will ship with the system an install object that contains the file-dependent indexes to the system entry point table. This object is the data management entry point table. The install object is a space object that contains a header and an entry for each type of file and device supported by the system.

The install object is created by common data management and accessed by common data management open to provide file redirection and device independence. It is also accessed when devices are defined. Addressability to the install object is provided through a pointer in the header of the DMCQ.

Figure DM-5 shows how this object fits with the system EPTAB (entry point table).

QDMEPTB

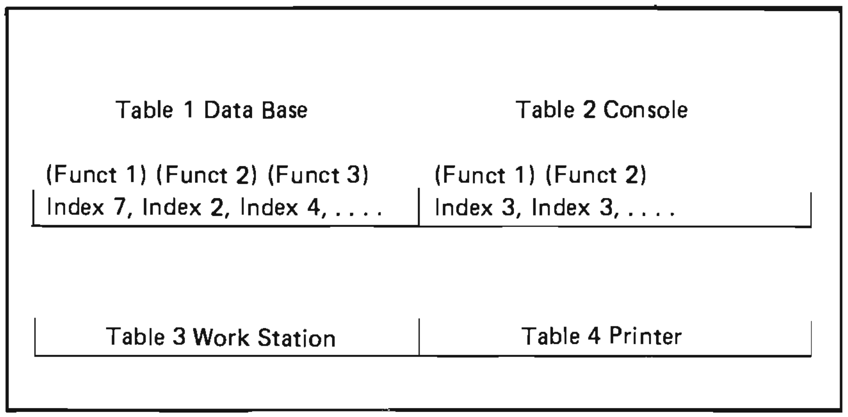


Table 1 Data Base
Table 2 Console
(Funct 1) (Funct 2) (Funct 3)
Index 7, Index 2, Index 4, ...
(Funct 1) (Funct 2)
Index 3, Index 3, ...
Table 3 Work Station
Table 4 Printer

The data management entry point table consists of multiple file-dependent tables. Each table contains indexes corresponding to data management functions (for example, put, get) and are used to index into QINSEPT(system entry point table).

	QINSEPT
1	SYS PTR
2	SYS PTR (QDBGETDR)
3	SYS PTR (QDMIFERR)
4	SYS PTR (QDBGETKY)
5	SYS PTR
6	SYS PTR
7	SYS PTR (QDBGETSQ)
8	SYS PTR
9	SYS PTR
A	SYS PTR

The system entry point table is an array of system pointers for all CPF modules.

Figure DM-5. Entry Point Table Structure

Data Management Communications Queue

During the start CPF process, work management invokes the common data management ?CRTDMQS macro to create the DMCQ (data management communications queue) and the MIRQ (machine interface request queue). The DMCQ is a temporary space object created inside the PAG (process access group). It is used to keep track of all the files opened or temporarily closed in the process, and to also keep track of devices passed to the process, file overrides that exist in the process, and common data management information for the process. The DMCQ consists of a header section and a number of fixed length entries that are chained together based on the type of entry.

The DMCQ header contains:

- A pointer to the PAG (used by common data management open)
- A pointer to the process control space
- A pointer to the install object containing EPTAB indexes
- A pointer to the PASA header
- A pointer to the data base logging control block
- A pointer to the data base space containing the list of UFCBs for the OPNDBF command
- An offset to open entries–not shared
- An offset to open entries–shared
- An offset to the user file override entries
- An offset to the spool file override entries
- An offset to passed device entries
- An indicator that the process is monitoring the I/O completion event for the console
- An indicator that the process is monitoring for the controlled cancel event

The type of entries that can be on the DMCQ are:

- Open files–shared ODPs
- Open files–nonshared ODPs
- User file overrides
- Spool overrides
- Passed devices

Machine Interface Request Queue

The MIRQ (machine interface request queue) is a machine interface queue used by all of the devices in a process that perform I/O. The queue is created extendable, with four initial entries and resides outside the PAG (process access group). The I/O feedback record is retrieved from the queue by a keyed dequeue. The key is 16 characters in length that contains:

- Component ID–char(2)
- Device name–char(10)
- Request I/O sequence number–char(2)
- Reserved–char(2)

Addressability to the MIRQ is provided by a pointer in the WCB (work control block).

Device Definition

The create device description commands create an LUD (logical unit description) and an associated space for each device. The LUD contains machine information, as defined by the machine. CPF materializes and modifies selected fields in the LUD. The associated space contains CPF-defined information. That information consists of two parts: common information and device-dependent information. The common information consists of a pointer to the process control space to which the device is allocated, obtain flags, the temporary close count, active session count, an offset (into the data management install object) to the index for the device, and so forth. The device-dependent information consists of a pointer to source/sink requests and source/sink data for passed data, the record format, and device suspended flag for display devices. For printer devices it contains the current print image, forms type, lines per inch to print, and so forth.

Device File Definition

The Create Device File command creates one space object consisting of two parts. The first *n* bytes are the inactive ODP (open data path) and the remainder of the object contains device file information.

Inactive Open Data Path

The inactive ODP (open data path) consists of the following:

Root Section

- Length of the inactive ODP
- Pointer to the buffer area
- Offset to lock list
- Offset to open feedback section
- Offset to I/O feedback
- Offset to spooling area
- Offsets to output BPCA (buffer processing communications area)
- Offsets to input BPCA
- Offsets to device file information
- ODP status bits
- Offset to DMCQ entry
- Open/close completion level
- Failing device number
- Source sequence number

Open Feedback Section

- File open count
- File type
- File Name
- Library name
- Spooling file number
- Device name and linkage list
- Overflow

I/O Feedback Section (Common)

- Offset to component-dependent section
- I/O statistics (number of gets, puts, and so forth)
- Record format name
- Device class (display)
- Device type (5251)
- Program device name
- Transaction identifier and data (80 bytes)

I/O Feedback Section (component-dependent)

Spoolable Device Spooling Area

- Variable area of the UFCB (user file control block) with parameters specified by the create device file command

Input BPCA

- Blocked record input

Output BPCA

- Blocked record output

Lock List For the File

The information contained in the device file consists of:

File attributes

- Image name
- Lines per inch

Record format descriptions

Note: The layout of the Data Base open data path is described in Section DB under *The Structure of Data Base Files.*

3270 Emulation

INTRODUCTION

The 3270 device emulation component allows the System/38 to appear as a 3270 Control Unit with attached devices to a remote host computer. When attached to a BSC 3270 network, the System/38 appears as a 3271 Model 2 Cluster Control Unit with attached devices. When attached to an SNA network, the System/38 appears as a 3270 controller with attached devices. The System/38 supports 50 emulation sessions for SNA, but only 32 for BSC.

System/38 emulates a 3270 Control Unit on SNA/SDLC lines. Using this support, a user at a 5250 Display Station on a System/38 may be connected to an application in a System/370 by nonswitched or switched PU2 lines.

System/38 emulates the 3271 Model 2 Control Unit on BSC multipoint lines. Using this support, a user at a 5250 Display Station on a System/38 may be connected to an application in a System/370.

System/38 emulates the 3277 Model 2 (1920 character) keyboard/display. In addition, it emulates the 3270 PF keys 13 through 24.

The 3284, 3286, and 3288 Printers are emulated as printers associated with the 3271 Model 2 Control Unit on BSC lines.

The 3284, 3286, 3287, and 3288 Printers are emulated as printers associated with the 3270 Control Unit on SNA lines.

GENERAL OVERVIEW

3270 emulation consists of two functions; display emulation and printer emulation. The interface with 3270 emulation is primarily by commands.

DISPLAY EMULATION MODULES

The 3270 emulation component consists of the following modules:

Note: Modules identified by an arrow (-->) are entry modules into the component. Indentation of a module shows its dependency on a preceding module.

-->QEM3270-Display Emulation Main Routine: This module starts 3270 emulation using parameters entered on the EML3270 command.

QEMWSEH-BSC Work Station Data Available Event Handler.

QEMSWSEH-SNA Work Station Data Available Event Handler: These modules handle most input entered by the user and pass it along to the host after translating the 5250 data streams into a 3270 data stream. It handles the HELP key and presents the HELP text. This module also allows the user to terminate the 3270 display emulation session.

QEMATTN-Attention Event Handler: This module handles the Attention Key Event (signaled when the Attention key is pressed) by unlocking the work station keyboard. This module can execute only when the process is not masked by another module. If the process is masked, the request is stacked and will execute when the process is unmasked.

QEMBSCEH–BSC Host Data Event Handler: This module receives 3270 data from the BSC host system, translates it to 5250 format, and sends it to the work station. It also recognizes 3270 Read Buffer and Read Modified commands, translates the contents of the 5250 Display Station to the proper 3270 response, and sends the response to the host system.

QEMSNAEH-SNA Host Data Available Event Handler: This module receives 3270 data from the SNA host system, translates it to 5250 format, sends it to the work station, and sends a positive response to the host. QEMSNAEH also recognizes 3270 Read Buffer, Read Modified, and Read Modified All commands, translates the contents of the 5250 display to the proper 3270 response, and sends the response to the host system. In addition, keeps track of the BIND/UNBIND status of the LU-LU session, keeps track of the ownership of the CD bit, sends the CD bit to the host when the host requests it, and sends negative responses to the host when errors are detected.

QEMTSTRQ-BSC Test Request Event Handler: This module handles the 5250 Test Request Key Event for BSC 3270 display emulation. It reads the work station screen, translates the 5250 data stream to a 3270 Test Request Read format, and sends it to the host.

QEMSYSRQ-SNA Test Request Handler: This module handles the 5250 Test Request Key Event for SNA 3270 display emulation. It switches the user back and forth between the SSCP-SLU and LU-LU sessions on the SNA device.

QEMSPEND-Suspend Emulation Display File Routine: This module is invoked by the work station function manager when the 3270 emulation display file is about to be suspended. It saves the contents of the work station screen.

QEMRST-Restore Emulation Display File Routine: This module is invoked by the Work Station FM when the 3270 emulation display file has been restored. It writes any new host data to the work station.

QEMIOERR-I/O Error Message Routine: This module is invoked by the display and printer emulation mainline routines when an I/O error occurs. It handles errors from the host, work station, and printer files and issues the correct error message.

QEMIEXIT-Invocation Exit Routine, Display: This module cleans up after display emulation when it is terminated by a cancel job, cancel request, or an escape message.

QEMWFCEH-Work Station Function Complete Event Handler: This module completes writing any data to the work station that was temporarily held up due to an extended work station function in progress (such as, handling the print key).

Figure EM-1 shows the Display Emulation operation.

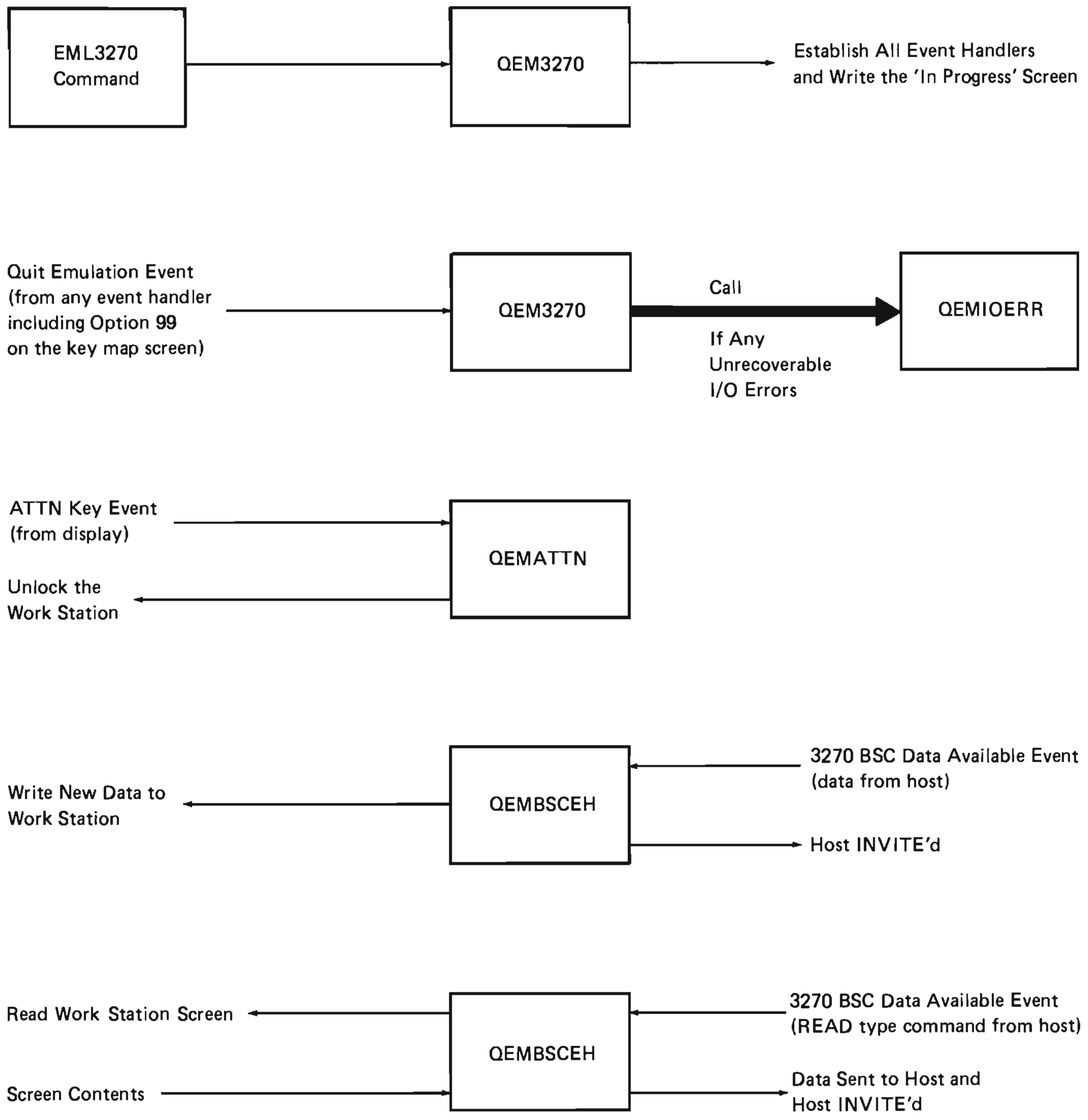

EML3270 Command
QEM3270
Establish All Event Handlers and Write the 'In Progress' Screen
Quit Emulation Event (from any event handler including Option 99 on the key map screen)
QEM3270
Call
If Any Unrecoverable I/O Errors
QEMIOERR
ATTN Key Event (from display)
Unlock the Work Station
QEMATTN
Write New Data to Work Station
QEMBSCEH
3270 BSC Data Available Event (data from host)
Host INVITE'd
Read Work Station Screen
Screen Contents
QEMBSCEH
3270 BSC Data Available Event (READ type command from host)
Data Sent to Host and Host INVITE'd

Figure EM-1 (Part 1 of 5). Display Emulation Overview

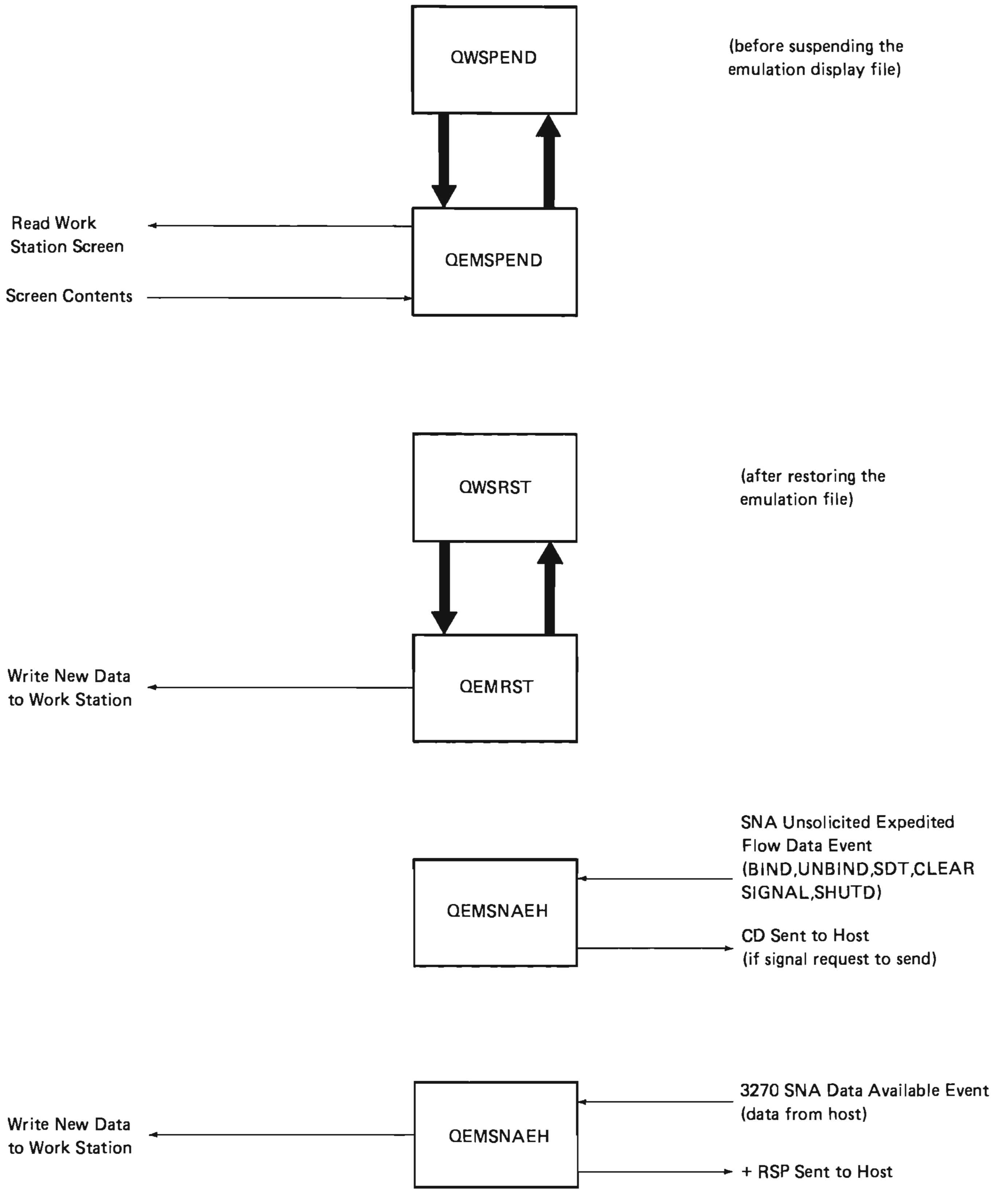

QWSPEND
(before suspending the emulation display file)
Read Work Station Screen
QEMSPEND
Screen Contents
QWSRST
(after restoring the emulation file)
Write New Data to Work Station
QEMRST
SNA Unsolicited Expedited Flow Data Event (BIND,UNBIND,SDT,CLEAR SIGNAL,SHUTD)
QEMSNAEH
CD Sent to Host (if signal request to send)
3270 SNA Data Available Event (data from host)
Write New Data to Work Station
QEMSNAEH
+ RSP Sent to Host

Figure EM-1 (Part 2 of 5). Display Emulation Overview

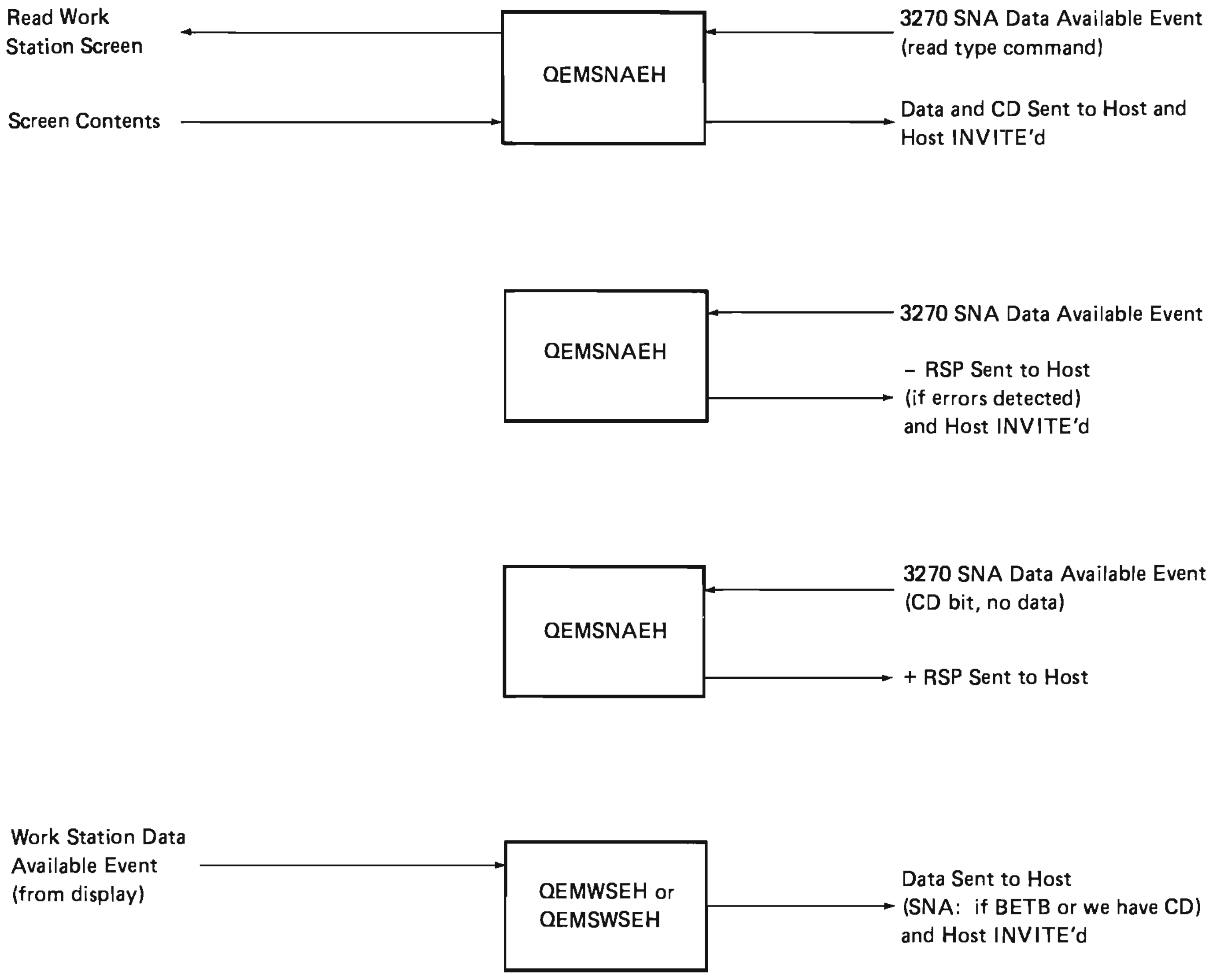

Read Work Station Screen
Screen Contents
QEMSNAEH
3270 SNA Data Available Event (read type command)
Data and CD Sent to Host and Host INVITE'd
QEMSNAEH
3270 SNA Data Available Event
– RSP Sent to Host (if errors detected) and Host INVITE'd
QEMSNAEH
3270 SNA Data Available Event (CD bit, no data)
+ RSP Sent to Host
Work Station Data Available Event (from display)
QEMWSEH or QEMSWSEH
Data Sent to Host (SNA: if BETB or we have CD) and Host INVITE'd

Figure EM-1 (Part 3 of 5). Display Emulation Overview

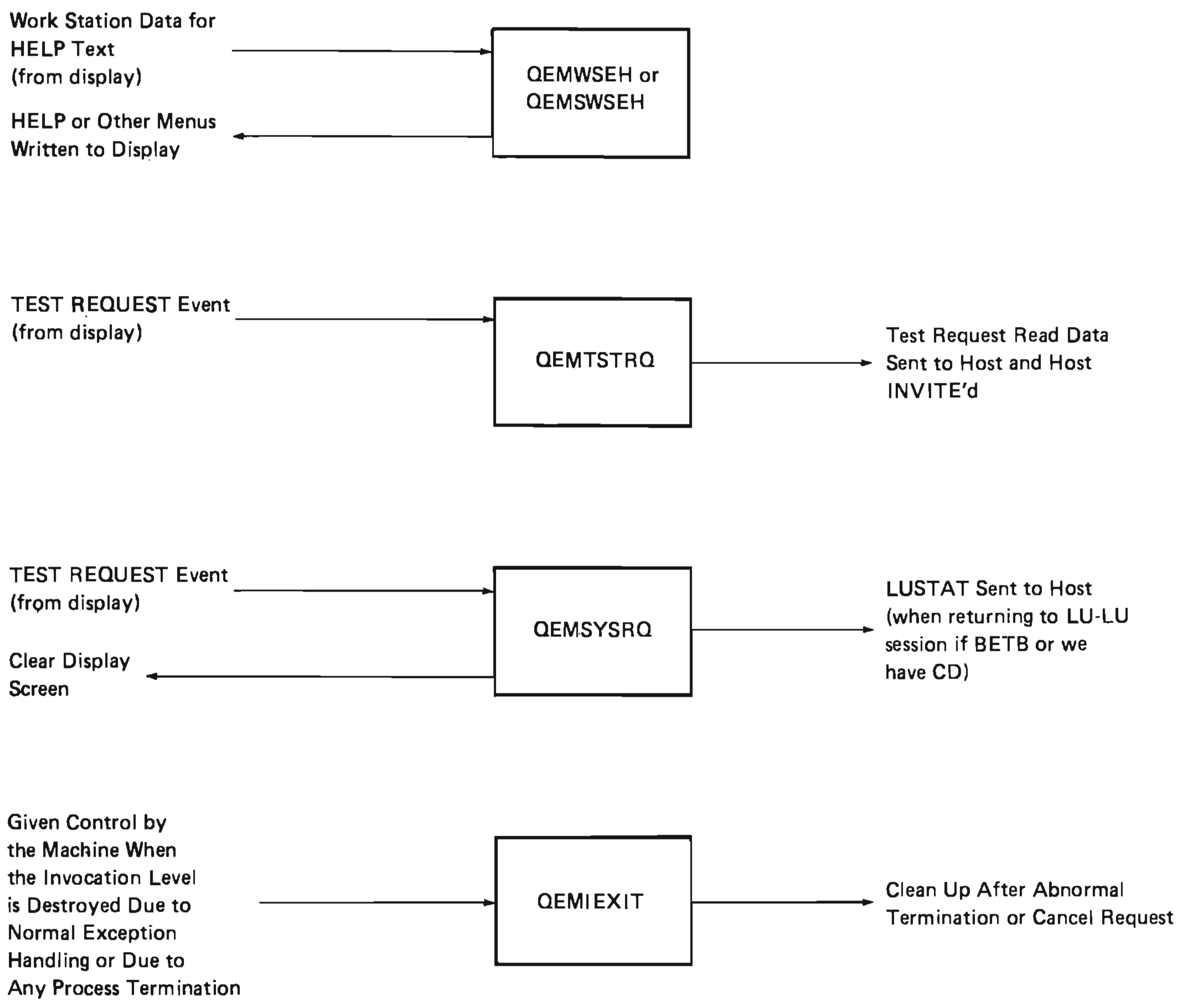

Work Station Data for HELP Text (from display)
HELP or Other Menus Written to Display
QEMWSEH or QEMSWSEH
TEST REQUEST Event (from display)
QEMTSTRQ
Test Request Read Data Sent to Host and Host INVITE'd
TEST REQUEST Event (from display)
Clear Display Screen
QEMSYSRQ
LUSTAT Sent to Host (when returning to LU-LU session if BETB or we have CD)
Given Control by the Machine When the Invocation Level is Destroyed Due to Normal Exception Handling or Due to Any Process Termination
QEMIEXIT
Clean Up After Abnormal Termination or Cancel Request

Figure EM-1 (Part 4 of 5). Display Emulation Overview

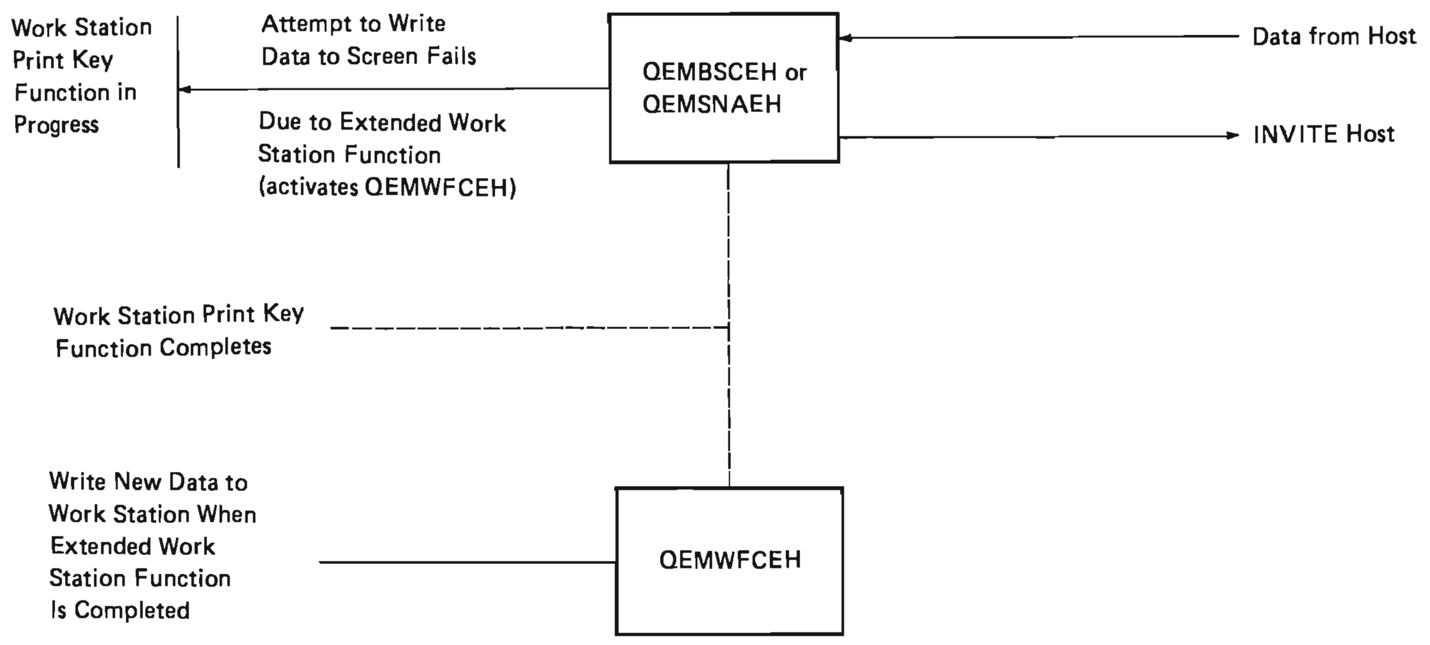

Work Station Print Key Function in Progress
Attempt to Write Data to Screen Fails
Due to Extended Work Station Function (activates QEMWFCEH)
QEMBSCEH or QEMSNAEH
Data from Host
INVITE Host
Work Station Print Key Function Completes
Write New Data to Work Station When Extended Work Station Function Is Completed
QEMWFCEH

Figure EM-1 (Part 5 of 5). Display Emulation Overview

Printer Emulation Modules

3270 Printer Emulation consists of the following Modules:

Note: Modules identified by an arrow (-->) are entry modules into the component. Indentation of a module shows its dependency on a preceding module.

-->QEMPESTR-Start Printer Emulation CPP: This module checks the input parameters and uses them to start printer emulation by one of the following methods:

- Transfer control to either the printer emulation BSC or SNA routines to request emulation in the current job.

- Submit a batch job request with the request data to call either the printer emulation BSC or SNA routines.

-->QEMPCNTL-Terminate Printer Emulation, Eject Emulation Output, and Emulate Printer Keys CPP: This module signals the terminate accept input event to the process that is performing printer emulation to end printer emulation, to eject the printer emulation output, or to emulate a PA1 or PA2 key.

-->QEMPEBSC-Printer Emulation BSC Routine: This module checks the existence of specified devices and files, initializes printer emulation, and repeatedly accepts input from the BSC file, translating the data to System/38 format. When the write control character indicates Start Print, QEMPRINT is called. If a 3270 read type command is received, QEMPEBSC sends the proper response to the host. When the terminate accept input event is received, QEMPEBSC performs the requested functions. On any type of termination request (TRMPRTEML or CNLJOB controlled), QEMPEBSC will close files and end printer emulation.

QEMPRINT-Printer Emulation 3270 Print Routine: This module is called by QEMPEBSC or QEMPESNA to print data sent by the host (in a 3270 data stream format).

-->QEMPESNA-Printer Emulation SNA Routine: This module checks the existence of specified devices and files, initializes printer emulation, and then accepts input from the communications file, translating the data to System/38 format. If the session type is LU-1, QEMPESCS is called to print the data. If the session type is LU3, QEMPRINT is called to print the data when the Write Control character indicates Start Print. If a read command is received during an LU-3 session, a negative response is sent to the host. When the CD bit is received during an LU-1 session, it is treated as a request from the host for a PA-1 or PA-2 key. If the terminate accept input event is received, QEMPESNA performs the requested function. When an I/O error is detected, QEMIOERR is called. On any type of termination request (TRMPRTEML or CNLJOB controlled), QEMPESNA will close files and end printer emulation.

QEMPESCS-Printer Emulation SCS Print Routine: This module is called by QEMPESNA to print the SCS data stream sent by the host.

QEMPESEH-3270 Printer Emulation SNA Event Handler: This module handles the SNA unsolicited data available event (expedited flow). The SNA commands that are handled by this module are BIND, UNBIND, SIGNAL, and SHUTD. The CLEAR and SDT commands are ignored.

QEMPEXIT-Invocation Exit Routine, Printer: This module cleans up after printer emulation.

Figure EM-2 shows the printer emulation overview.

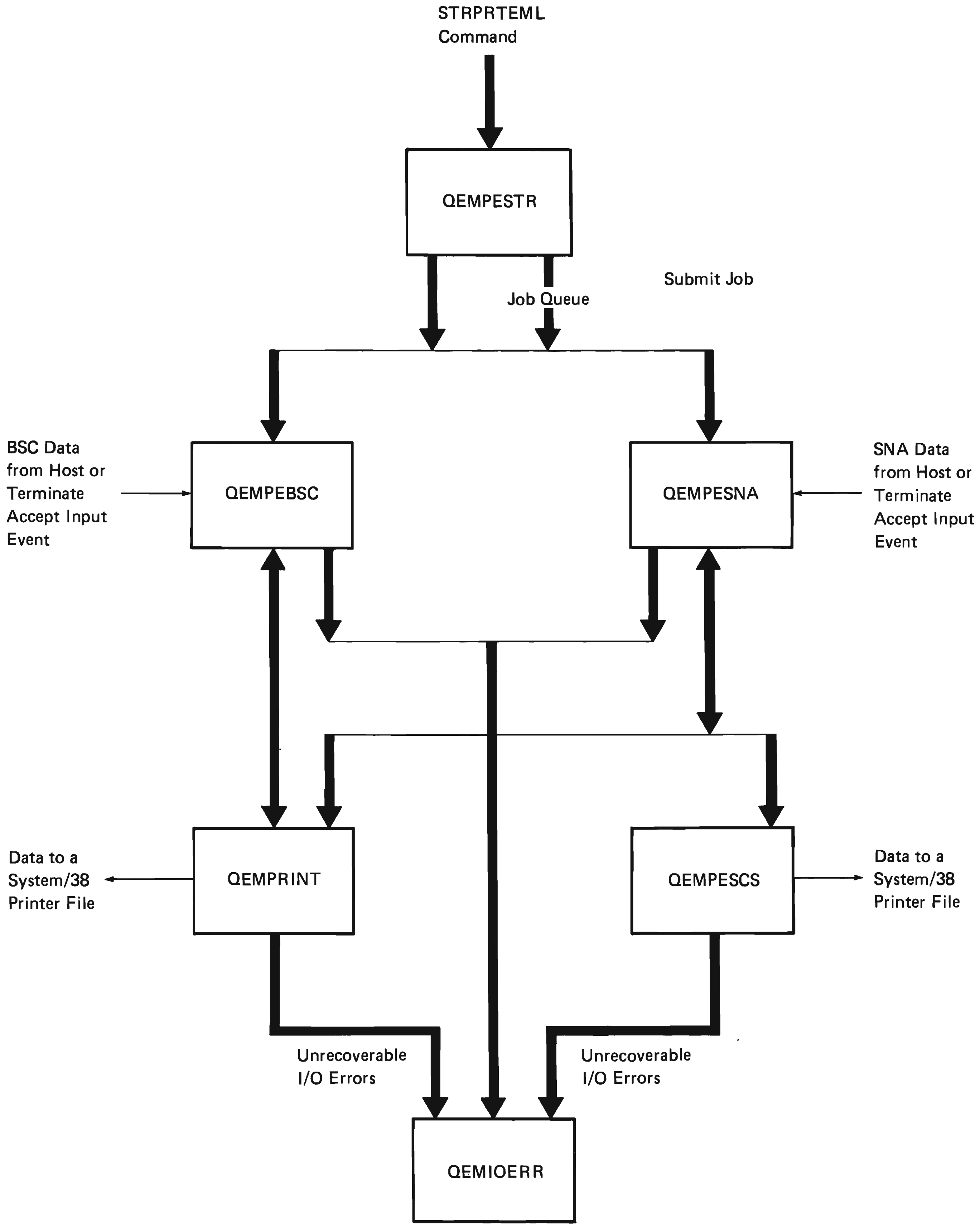

STRPRTEML Command
QEMPESTR
Submit Job
Job Queue
BSC Data from Host or Terminate Accept Input Event
QEMPEBSC
QEMPESNA
SNA Data from Host or Terminate Accept Input Event
Data to a System/38 Printer File
QEMPRINT
QEMPESCS
Data to a System/38 Printer File
Unrecoverable I/O Errors
Unrecoverable I/O Errors
QEMIOERR

Figure EM-2 (Part 1 of 2). Printer Emulation Overview

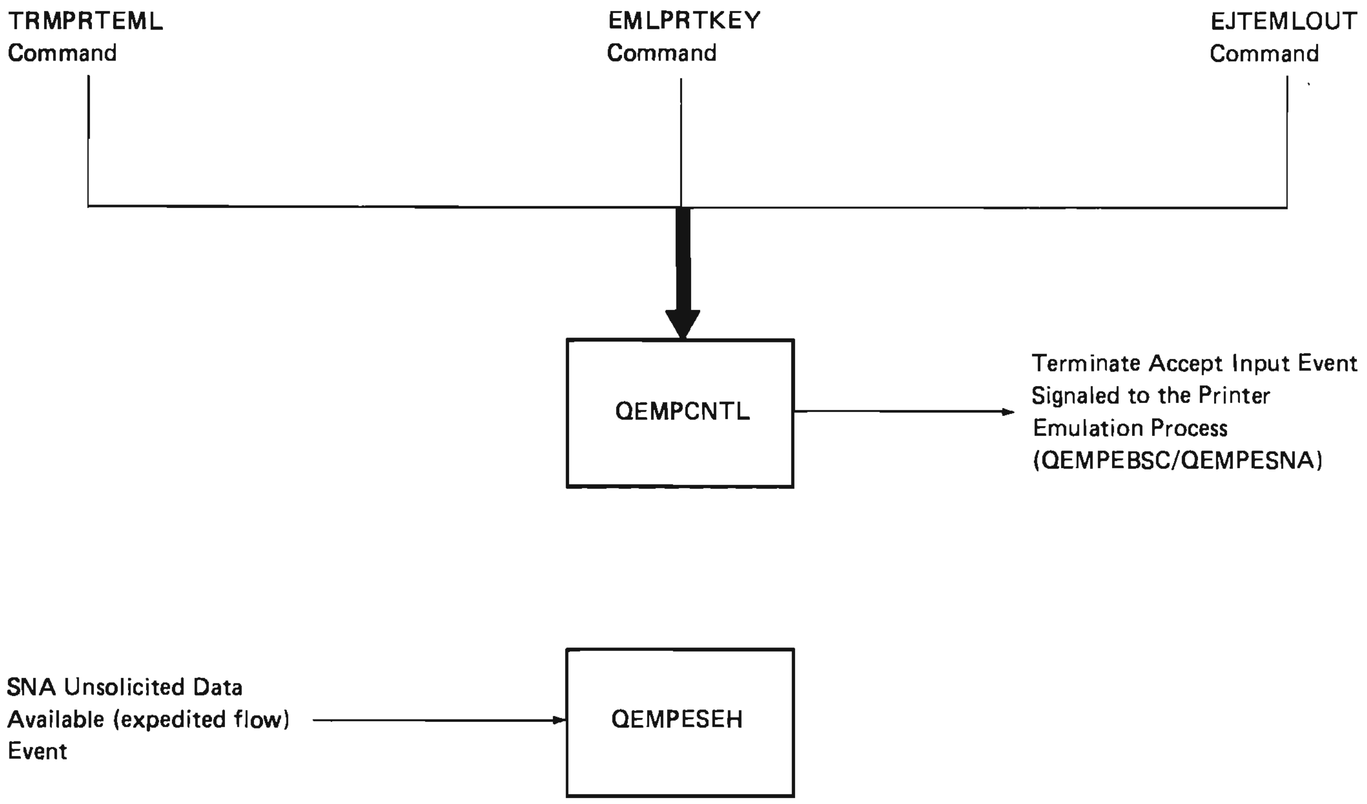

TRMPRTEML Command
EMLPRTKEY Command
EJTEMLOUT Command
QEMPCNTL
Terminate Accept Input Event Signaled to the Printer Emulation Process (QEMPEBSC/QEMPESNA)
SNA Unsolicited Data Available (expedited flow) Event
QEMPESEH

Figure EM-2 (Part 2 of 2). Printer Emulation Overview

INTRODUCTION

The Finance Support (FN) component of CPF provides for the attachment of 4701 and 3694 finance control units to the System/38 on an SNA/SDLC communications link. The communications link can be point-to-point or multipoint and can be switched or nonswitched.

The 4701 and 3694 control units can share the same communications line with APPC sessions, 5251 Model 12, 5294, and 3274 remote control units. Control unit descriptions can be created for the 4701 and 3694. Device descriptions can be created for the 4704, 3624, and 3694.

The Finance Support offers a choice of interfaces:

- Submit Finance Job command (SBMFNCJOB)
- Finance I/O Manager (FIOM)
- User Defined Data Stream (UDDS)
- System/38 3270 SNA Remote Attach Support

GENERAL OVERVIEW

The following modules make up the Finance Support component.

-->QFNMNTBL–Manage Tables: This is the command-processing program invoked for any of the Finance Support manage table commands (MNGDEVTBL, MNGPGMTBL or MNGUSRTBL). It also handles the table selection display, allowing users to select a table to be added, updated, or removed. When a new table name is entered or when an existing table name is selected for update, QFNMNTBL invokes QFNMNMBR to display the requested table.

QFNMNMBR–Manage Table Member: This program is responsible for managing a specific finance device, or program, or the user table requested for update by any of the manage table commands (MNGDEVTBL, MNGPGMTBL or MNGUSRTBL). It displays table entries currently defined and provides additional fields for input of new entries.

-->QFNSBMJB–Submit Finance Job (SBMFNCJB): This is the command-processing program for the SBMFNCJOB command. This program verifies that the finance device, or program, and the user tables specified as parameters on the SBMFNCJOB command do in fact exist, and that the user is authorized to objects given as command parameters. It then submits a batch job to the QFNC subsystem. This job calls QFNROUTE to establish and manage communications with the finance devices, as explained below.

QFNROUTE–Finance Router: This program provides a router function for the SBMFNCJOB interface. It acquires devices specified in the finance device table, invites those devices, accepts and verifies user IDs received with SNA INIT-SELFs, and calls the requested System/38 application programs to process financial transactions. Upon receiving a TERM-SELF, this module releases and then reacquires the device requesting session termination, allowing a new session to be established with that device.

Finance I/O Managers

The following modules constitute FIOM support; these modules are the Finance I/O Managers. They are provided as an alternative to UDDS communications for those Finance Support users who desire direct communications between their System/38 transaction-processing programs and the finance control unit application.

Note: An arrow (-->) identifies the modules that can be called as external subroutines of the user's application program.

-->QFNWRT–Write: This program will allow users to write up to 512 bytes of application data to a specified finance device. The user supplies as input to QFNWRT an indicator relating the type of data to be written, the length of that data, the data itself, and the finance device to which to write. QFNWRT accepts these input parameters and invokes QFNIOMGR to handle the I/O operation.

-->QFNWRTI–Write Invite: This program will write up to 512 bytes of application data to a specified finance device in a manner similar to that of QFNWRT. In addition, QFNWRTI will invite the device for communications. This program should be used in conjunction with the program QFNREAD or QFNREADI to allow reading of the data received as a result of the write invite issued to the finance device.

-->QFNREAD–Read: Reads up to 512 bytes of data from a specified finance device. Prior to calling QFNREAD, the device must be invited by the program QFNWRTI. The user supplies as input to QFNREAD the name of a device from which to read. This program calls QFNIOMGR to perform the read operation, then returns to the user's application program the data received, along with the length and the type of that data.

-->QFNREADI–Read From Invite: This program will read up to 512 bytes of data from any invited finance device. Prior to calling QFNREADI, the devices must be invited by the program QFNWRTI. This module functions in a manner similar to that of QFNREAD, with the name of the finance device, from which data is received, returned as an additional output parameter to the user's application program, rather than supplied as input to QFNREADI.

QFNIOMGR–Finance I/O Manager: This is the primary finance I/O manager. It acts as an external subroutine of QFNREAD, QFNREADI, QFNWRT and QFNWRTI. This program accepts any input parameters supplied by the user and performs the requested input or output operation to the finance devices.

Submit Finance Job (SBMFNCJOB) Command Interface

Figure FN-1 and the following text describe the module flow for the SBMFNCJOB interface.

1. QFNSBMJB is the command processor for the SBMFNCJOB command. This module submits a batch job that calls QFNROUTE to establish and manage communications with finance devices.

2. QFNROUTE provides a router function for the SBMFNCJOB interface. It acquires and releases devices, and handles communications between your System/38 transaction processing programs and the finance control unit application.

3. QFNMNTBL is the command processor for the Manage Table commands. It handles the table selection display and invokes QFNMNMBR when a specific table is selected.

4. QFNMNMBR manages the specific device program, or the user table selected for display or update.

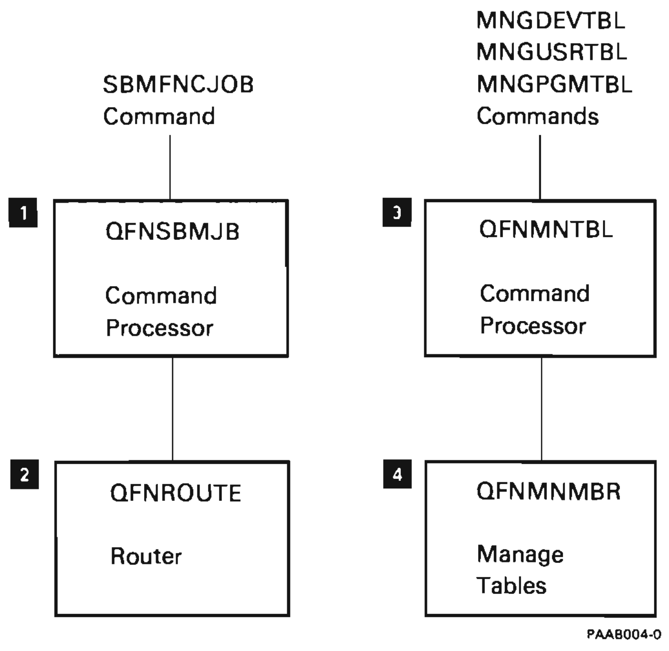

SBMFNCJOB Command
MNGDEVTBL MNGUSRTBL MNGPGMTBL Commands
1 QFNSBMJB Command Processor
3 QFNMNTBL Command Processor
2 QFNROUTE Router
4 QFNMNMBR Manage Tables
PAAB004-0

Figure FN-1. Submit Finance Job Interface

Finance I/O Manager Interface

Figure FN-2 and the following text describe the Finance I/O Manager interface.

1 QFNWRT accepts user parameters such as devices, data type, data length, and the data itself, and then invokes QFNIOMGR to handle I/O operations.

2 QFNWRTI handles writes similar to QFNWRT. In addition, it invites the device for communications.

3 QFNREAD reads data from a finance device. The device must be previously invited by QFNWRTI. The user program supplies the name of the device.

4 QFNREADI reads similar to QFNREAD except that the name of the device, from which data was received, is returned to the user program.

5 QFNIOMGR acts as an external subroutine for the read and write commands. It performs the requested I/O operations to finance devices.

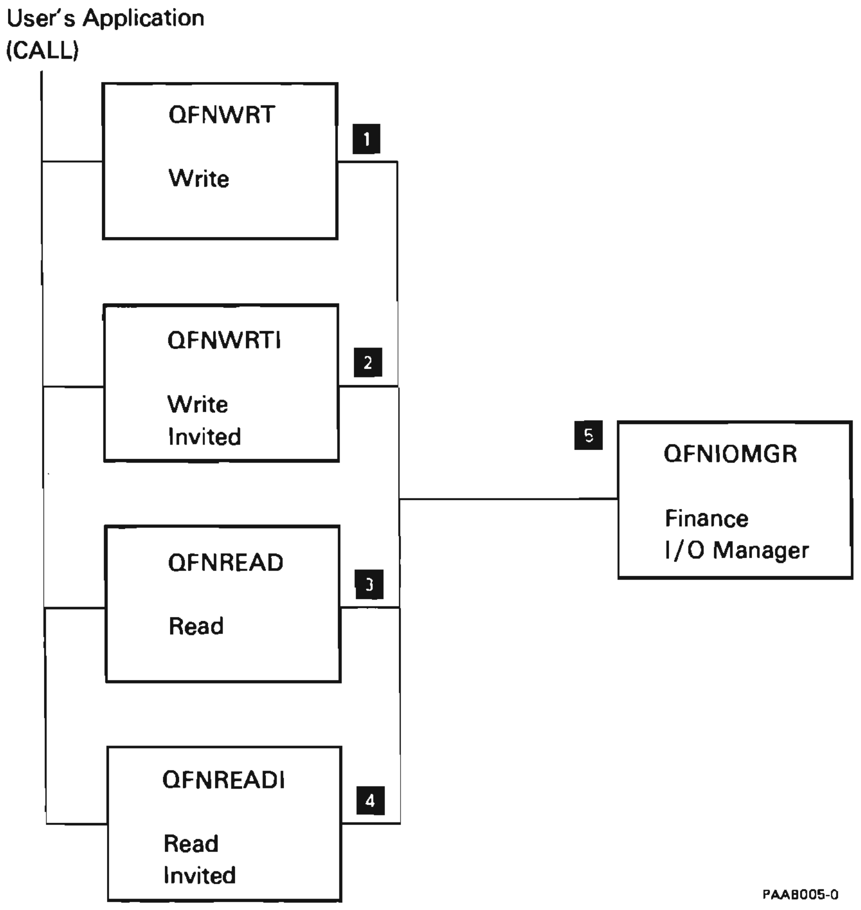


Figure FN-2. Finance I/O Manager Interface

INTRODUCTION

CPF graphics allows the user to add color and pictures to application programs. High level language programs call CPF graphic routines to help construct the pictures. Each of the routines is like a small, self-contained program. The routines are organized in two groups:

- Graphic data display manager routines (GDDM)
- Presentation graphics routines (PGR)

GDDM Routines

GDDM routines perform basic graphic tasks, such as drawing a line from point A to point B. A series of these line drawing routines in an application program can produce a more complex picture. Also, GDDM routines are called in an application program to perform such tasks as initializing and terminating the graphics environment, defining characteristics for functions that other GDDM routines will perform (such as setting the color and width of a line that another GDDM routine will draw), and sending the picture to the work station.

PGR Routines

PGR routines provide a fast and efficient way to convert numeric data into color charts in an application program. One PGR routine will specify the type of chart used to present the data, while other PGR routines will label the data and specify chart headings.

PGR routines are built with sets of GDDM routines. An application can have any mixture of GDDM and PGR routines.

GENERAL OVERVIEW

Graphics Modules

Figure GD-1 and the following text show the structure of the graphics component.

1 The purpose of the application interface (AI) is to provide the interface between the user and GDDM. This includes:

- Conversion of application calls to an internal format
- Invocation of the appropriate GDDM unit to process application calls
- Building and sending of error messages
- Invocation of specified user error exit
- Initialization
- Normal termination

The application interface is the top layer within GDDM and PGR. All GD and entry point PGR modules are in this unit.

2 The full screen manager contains the subcomponents responsible for graphic data manipulation and control.

3 The presentation graphics routines are a set of routines that access the GDDM primitives but allow a higher level interface. For example, a single PGR routine will draw a VENN diagram by calling several other GDDM routines.

4 The terminal services interface is a subset of GDDM that provides subsystem/device dependent functions such as I/O.

5 The common services interface (CSI) performs general functions such as allocation and deallocation.

6 The environmental services interface (ESI) is a subset of GDDM that provides subsystem dependent control functions such as storage management.

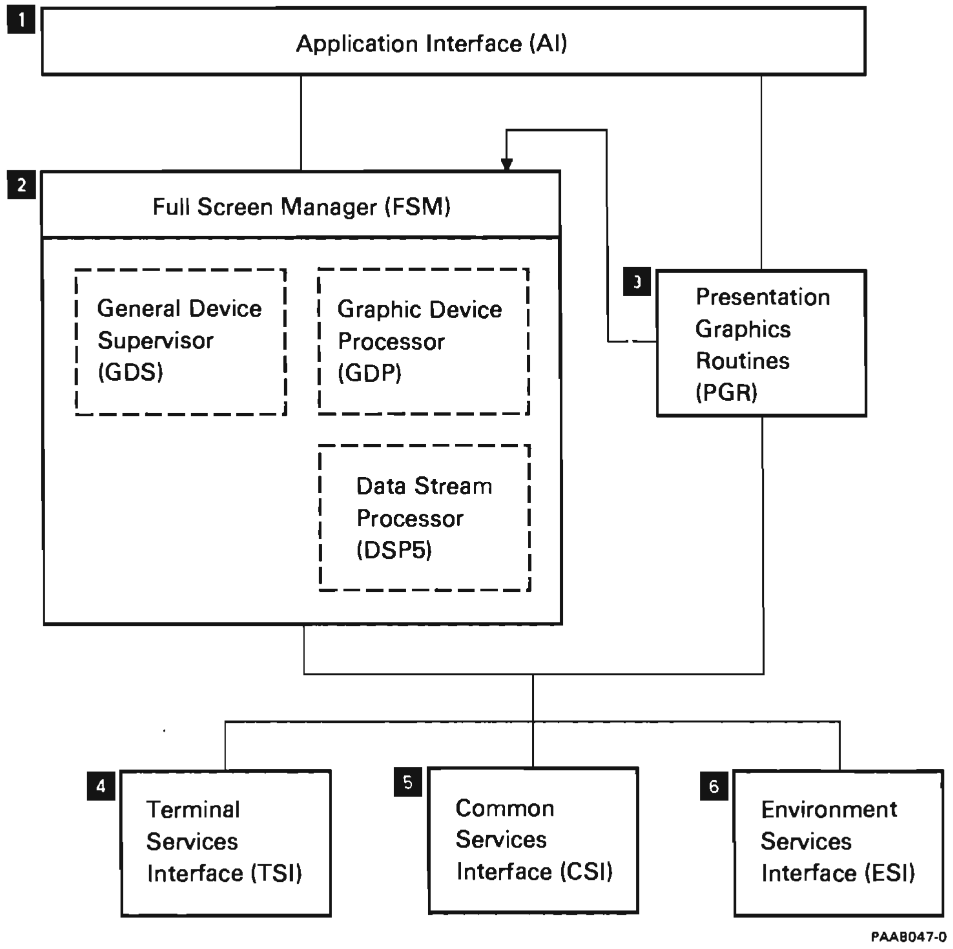

1
Application Interface (AI)
2
Full Screen Manager (FSM)
General Device Supervisor (GDS)
Graphic Device Processor (GDP)
Data Stream Processor (DSP5)
3
Presentation Graphics Routines (PGR)
4
Terminal Services Interface (TSI)
5
Common Services Interface (CSI)
6
Environment Services Interface (ESI)
PAA8047-0

Figure GD-1. Graphics Component Structure

AI Modules

-->QGDACFP–Control Function Processor: This module is the control function processor for the application interface and routes control to the appropriate AI module based on the RCP code.

-->QGDACIN–Initialization: This module creates a graphics control space object and storage space object. The control blocks in the control space are initialized.

A scope message is sent to the first invocation below the request level processor/receiver or to the first invocation in the current request level if no processor/receiver exists. This scope message will cause the scope message handler module, QGDASCPH, to be invoked if the target invocation of the scope message is removed from the invocation stack.

-->QGDACTRM–Termination Function: This module terminates a graphics instance. The exact processing depends on the reason that termination is invoked.

-->QGDACO–Controller Router: This module handles all GDDM and PGR application calls. The application program codes calls to the GDDM entry modules, such as FSINIT, and GSLINE.

When an entry module has been called by an application program, it in turn calls QGDACO, passing all the user-supplied parameters.

-->QGDAEP–Error Processor: This module handles the error notification and feedback. It is called to process the graphics function FSEXIT and FSQERR. It is also called to signal an error that has occurred somewhere in graphics processing.

-->QGDAINVP–Invoke User Error Exit Program: This module transfers control to the specified user error exit program.

When an error occurs in GDDM, the user is notified via diagnostic messages. If the user has specified an error exit, with FSEXIT, GDDM will also give control to that module with sufficient information to allow the program to perform processing based on the particular error. The purpose of this module is to invoke that program and pass a control block with the error information.

-->QGDASCPH–Scope Message Handler: This module ensures the cleanup of GD referenced objects and spaces.

PGR Modules

-->QGDBADTM–Datum Reference or Datum Line: In state 1, this module invokes ADMBSET to store the datum reference. In state 2, this module draws the datum line.

-->QDBARS–Draw Bar Graphs and Place Values: This module draws a bar graph.

-->QGDBASEL–Select the Current Axis: This module processes both CHXSEL and CHYSEL calls.

-->QGDBATT–Set the Current Attributes: This module calls general graphics to set the attributes to the desired value.

-->QGDBBGS–Business Graphics Supervisor: This module is the main routine for the reentry portion of PGR. It is entered by the PGR entry modules and after some preliminary processing invokes the appropriate procedure.

-->QGDBBLNK–Blank Area Under Character String: This module shades an area whose boundaries are supplied by the parameters with a solid background shading pattern so that the characters are more legible.

-->QGDBCHRT–Route to Plotting Routine: This module will draw the axis and then draw the type of chart that was requested.

-->QGDBCHSG–Set the Character Attributes: This module sets the current color, mode, set number, and multiplier.

-->QGDBCHVU–Set the Viewport and Window: This module sets the viewport and window based on the input parameters.

-->QGDBCRNG–Set Range Values for Autoscaling: This module determines if the range has been set to either axis. The minimum and maximum values are passed as parameters. After setting the range, the axis will be drawn.

-->QGDBDKEY–Legend Construction: This module constructs a legend of specified format at a specified position within the chart area.

-->QGDBDOAX–Draw Component: This module draws axes, tick marks, grids, datum lines, reference lines, and titles.

-->QGDBDRAW–Draw Heading and Determine Plot Boundaries: This module sets processing to state 2 and draws the headings for the chart.

-->QGDBDRAX, QGDBDSAX, QGDBDSDO–Draw Complete Axes, Titles, and Labels: Draw all components of the plot area other than the plot itself. This includes axes, axis title, labels, tick marks, datum reference lines, and grid lines.

-->QGDBDTTL–Draw Axis Title: This module determines the position for a title of either a vertical or horizontal axis and draws it.

QGDBEDTX–Edit Text Strings

-->QGDBGFMT–Generate Format for Numeric Labels: This module determines the number of integer positions, fraction positions, sign, and if necessary, E-format positions required for conversion to EBCDIC and display of a label.

-->QGDBGFTX–Get/Free Storage for Text Strings: This module frees existing storage, obtains needed storage, and fills the storage with the text string and header information.

-->QGDBGLBS–Generate Labels: This module generates the EBCDIC representation of an input value.

-->QGDBHIST–Draw Histogram

-->QGDBLABL–Label an Axis: This module defines the labeling parameters according to the type of labels required by the specifications and parameters received. It then draws each label next to the axis.

-->QGDBMAX–Determine Minimum and Maximum Axis Values: This module finds the minimum and maximum values passed in the array by the calling program. If the array is composed of multiple components, then the maximum value will be the sum of the individual items for that component.

-->QGDBMOVE–Process Draw Requirements for Charts

-->QGDBNOTE–Chart Annotation: This module constructs the required notation at the location specified by the position code and the note offset values.

-->QGDBPIE–Draw Pie Chart

-->QGDBPLOT–Draw Line and Surface Charts

-->QGDBRNIT–Set PGR Control Blocks

-->QGDBSET–Set Values in Control Blocks

-->QGDBSTRT–PGR Supervisor

-->QGDBVENN–Draw Venn Diagram

FSM Modules

-->QGDDBCRT–Create Partition Set

-->QGDDBDEF–Create Default Partition Set

-->QGDDBDEL–Delete Partition Set

-->QGDDBFN1–Process Partition Calls

-->QGDDBSEL–Select Partition Set

-->QGDCAS–Allocate Symbol Set Table Entry: This module is part of the FSM common device processor (CDP). It is called by the load/define symbol set modules to allocate a suitable entry in the symbol set table.

-->QGDDCCD–Convert Call Definitions: This module is called to convert call images from one format to another.

-->QGDDCDS–Release Symbol Set: This module handles the release symbol set functions. It is one of the main entry points of the common device processor. It is invoked by the main supervisor router, and processes the GSRSS routine.

-->QGDDCES–Release Symbol Set: This module is called by the load/define symbol set modules and by the delete symbol set module to release a symbol set. It is also called by the graphics device processor to release a symbol set that is no longer required for graphics.

-->QGDDCGS–Load/Define Graphics Symbol Set: This module processes the GSLSS routine.

-->QGDDCOS–Get Symbol Set: This module handles an internal request to get access to symbol set definitions. It satisfies the request either by invoking ESI to read the symbol set or it may remember that the required symbol set is one of the most recently queried. In this case, this module already has access to it.

-->QGDDCPC–Page Control: This module handles most of the page control routines.

-->QGDDCPU–Query Unique Page: This module is invoked to process a query unique page number request. This returns a page number that is not currently in use.

-->QGDDCQD–Query Device Characteristics: This module handles the FSQDEV routine.

-->QGDDCQS–This module handles all the query symbol set functions of the GDDM FSM.

-->QGDDCRS–Read/Write Symbol Set

-->QGDDCTE–Initialize/terminate

-->QGDDCVS–Validate Symbol Set: This module validates a set of symbol set definitions passed to it.

-->QGDDCWIN–Control Page Window

-->QGDDECRT–Create Partition: This module creates a new partition block.

-->QGDDEDEF–Create Default Partition

-->QGDDEDEL–Delete Partition

-->QGDDEFN1,2–Process Partition Functions

-->QGDDESEL–Select Partition

-->QGDDGAR–Buffer Manager

-->QGDDGCE–Arc Simplification: This module breaks arcs into a monotonic arc.

-->QGDDGCL–Line Clipping: This module clips a line to the viewport boundaries.

-->QGDDGCR–Correlation Module

-->QGDDGGI–Query Input Device Data

-->QGDDGIG–IDF Generator: This module generates IDF orders to set the attributes of a graphic primitive.

-->QGDDGIO–Initialization: This module controls the initialization and termination of the GDP. Initialization is triggered internally by the GDP when the graphics field is created. Termination is invoked by DSCLS.

This module is responsible for the loading and unloading of the dependent modules and their initialization and termination.

-->QGDDGI5–5292 Display Initialization: This module handles initialization, field creation, termination, and field deletion for the 5292 Model 2 display.

-->QGDDGI6–Plotter Initialization: This module handles initialization, field creation, termination, and field deletion for the 737x plotter.

-->QGDDGMM–Matrix Multiplier: This module multiplies two matrices of given dimensions.

-->QGDDGPA–Primitive Attribute Module: This module handles color, line width, paint, pattern, character mode, set, box, angle, and direction.

-->QGDDGPC–Character String Module: This module processes GSCHAP, GSCHAR, and GSQTB character string calls.

-->QGDDGPE–Arc Primitive Module: This module handles arc requests. Arcs may be specified by GSARC, GSELPS, and GSPFLT. The external form of these functions is processed and a common GDF order is generated from them.

-->QGDDGPI–GDF Interpreter: This module interprets a GDF string.

-->QGDDGPM–Image Processor: This module processes image calls including GSIMG and GSIMGS.

-->QGDDGPO–GDF Exporter: This module returns GDF data to the application. This includes the handling of the GSGETS, GSGET, and GSGETE functions.

-->QGDDGI4–Printer Graphics Initialization: This module handles initialization, field creation, termination, and field deletion for printer graphics.

-->QGDDGP1–GDF Generator: This module accepts GDF orders and saves them for GDF retrieval.

-->QGDDHI06–737x Plotter Table Builder: This module is called by QGDDGI6 to return the graphics default module index for plotter support.

-->QGDDHI04–522x, (10 CPI) Printer Table Builder: This module is called by QGDDGI6 to return the graphics default module index for 522x, printer support.

-->QGDDHI14–4214 Printer Table Builder: This module is called by QGDDGI6 to return the graphics default module index for 4214 printer support.

-->QGDDHI24–15 CPI Printer Table Builder: This module is called by QGDDGI6 to return the graphics default module index for printer support.

-->QGDDGCTF–Color Table Definition Selection Function.

-->QGDDGS1–Display Segment Buffer Manager for producing GDF.

-->QGDD1C–Data Stream Processor for Dummy Device Support.

-->QGDDGPR–Primitive Operations: This module, which is one of the entry points to the GDP, handles moves, line, vector, marker, area, and end area calls.

-->QGDDGPS–Default Picture Space: This module calculates a default picture space and converts it into 32 K bytes coordinates.

-->QGDDGP5–Data Stream Generator: This module accepts GDF orders and generates 5292 Model 2 display data stream orders from them.

-->QGDDGP6–737X Plotter Data Stream Generator: This module accepts GDF orders and generates 737X plotter (IBM-GL) data stream orders from them.

-->QGDDGQC–Quickcell Routine: This module allocates and frees storage for bundles of control blocks.

-->QGDDGQI–Query Character Spacing: This module, which is called internally, computes the relative coordinates of the bottom right and top left corners of a character box by considering current character angle, mode, direction, and box attributes.

-->QGDDGSE–Segment Operations Module: This module, which is one of the entry points to the GDP, handles segment creation, closure, deletion, and clear calls. It is also invoked by the CDP to handle page deletion.

-->QGDDGSQ–Segment Querying Module: This module, which is one of the entry points to the GDP, handles the following calls: GSQCUR, GSQMAX, GSQCEL, GSQCLP, and GSCLP.

-->QGDDGS5–5292-2 Display Segment Buffer Manager: This module manages (creates, allocates, deallocates, and destroys) segment buffers. A segment buffer is the buffer into which the data stream generators (QGDDGP5 and QGDDGP6) place actual device graphics orders.

-->QGDDGWI–Window Definition: This module, which is one of the entry points to the GDP, handles segment window defining, viewport defining, picture space defining, picture space querying, and graphics field defining.

-->QGDDGXC–Character String and Marker Expansion: This module expands a character string or marker order into a series of lines.

-->GQDDGXE–Arc Expansion: This module expands circular and elliptic arcs to a series of straight lines.

-->QGDDHI05–5292-2 Display Table Builder: This module is called by QGDDGI5 to return the graphics default module index.

-->QGDDM–AIC Alternate Entry Point: This module serves as an entry point to GDDM/PGR, which can be called with parameters to indicate the function required.

-->QGDDSDQ–Query Reply Processor: Provides a common service to the processing modules for decoding a query reply.

-->QGDDSDS–Device Services: This module handles all device services (DS....) calls to FSM. This module interfaces with TSI adapters for device initialization and termination.

-->QGDDSEH–Error Handler: This module handles an error detected by any GDDM GSM module.

-->QGDDSF1–Device Family 1 Processor: This module performs family specific processing for DSOPEN, DSCLS, and DSRNIT.

-->QGDDSOO–Split Open Option: This module converts between external parameters of DSOPEN and DSQDEV (processing options and name list), and the internal option lists.

-->QGDDSRO–Control Router: This module is the only entry point to GDS and FSM. This module controls the routing of external calls to the appropriate FSM processor and performs basic processing for initialization and termination.

-->QGDD5C–Data Stream Processor: This module builds the data stream buffers for output to the device. It appends all control commands for the start and the end of the graphics mode as well as all control commands required for each buffer sent to the device.

ESI Modules

-->QGDEABND–Abend Termination: This module processes the abend conditions for GDDM.

-->QGDEOSDO–Load/release GSS: This module is called to get or release a graphics symbol set.

-->QGDEROOF–Router and Storage Manager

TSI Modules

-->QGDILASP–LUD Initialization: This module initializes a portion of the GD device dependent section of a display LUD associated space.

This module is called by the DC component during vary-on processing for 5292 Model 2 devices.

-->QGDLAR1F–Acquire/release buffers: This module acquires and releases buffers required by TSI.

-->QGDLIN1F–TSI Initialization: This module initializes the GD component for a particular device. Each device used has its own control blocks. This module processes the DSOPEN routine.

Initialization functions include validity checking the DSOPEN options, locking the specified device, opening the QDGDDM display file, error handling and back-out, and terminal query functions.

-->QGDLRN1F–Reinitialization: This module reinitializes GDDM for a particular device. This occurs when the DSRNIT or FSRNIT routines are called. For FSRNIT, GDS generates internal DSRNIT requests for each open device.

-->QGDLRO00–Dummy Device Router: This module serves as the router for GDDM requests to dummy devices.

-->QGDLRO1F–Function Module and I/O Services: This module routes all TSI requests and performs I/O processing.

-->QGDLTM1F–Termination Function

-->QGDLTQ00–Query Device Function: This module returns information about the device.

-->QGDNUMER–Numerical Preprocessor: This module generates smooth curves.

CSI Modules

-->QGDYGQC–Quickcell Routine: This module allocates and frees storage for bundles of control blocks.

-->QGDYINTM–Initialization/termination: This module provides initialization and termination for the common services interface (CSI).

-->QGDYRO00–CSI Router: This module routes the CSI request to the appropriate module.

-->QGDYRSRL–CSI Reserve/Release Resource Handler

-->QGDYTRIG - CSI Trigonomic Functions: This module returns the sine and cosine for the specified angle.

INTRODUCTION

The installation component of CPF (control program facility) is responsible for installing and initiating CPF on the IBM System/38.

To invoke the installation component, an operator performs an AIPL (alternative initial program load).

GENERAL OVERVIEW

Installation Modules

The installation component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

- -->QINIT–Installation Loader: This module loads QINSTALL into the system when the installation is from diskette.
  - QINSTALL–Stand-Alone Portion of Installation: This module loads all CPF objects except for subsystem descriptions and data base files. It also initiates the initial CPF process.
  - QINCPF–Initialization Requiring CPF: This module is executed during the start CPF process. It controls I/O reconfiguration, the loading of subsystem descriptions and data base files, and controls any release/modification-dependent initialization. To control the I/O reconfiguration, a prompt is sent to the system console asking if the I/O descriptions on the save/restore medium are to be restored. QINRIO is called if the response is affirmative.
    - QINRIO–Restore I/O Configuration: This module deletes all existing I/O descriptions except for the system console and save/restore device(s). The save/restore component is called to restore the I/O descriptions from the save/restore medium.
      - QINRIOOH–Restore I/O Description Object Handler: This module, after initial processing by the save/restore component, receives the I/O description data and creates the appropriate logical unit description, control unit description, or network description.
  - QINFIXUP–Release Dependent Initialization: This module creates job and output queues and grants private authorities. In addition, this module performs other functions that are dependent upon the specific CPF release/modification level.
- -->QINITT–Installation Loader: This module loads QINSTALL into the system when the installation is from magnetic tape.
  - QINSTALL–Stand-Alone Portion of Installation: This module loads all CPF objects except for subsystem descriptions and data base files. It also initiates the initial CPF process.
  - QINCPF–Initialization Requiring CPF: This module is executed during the start CPF process. It controls I/O reconfiguration, the loading of subsystem descriptions and data base files, and controls any release/modification-dependent initialization. To control the I/O reconfiguration, a prompt is sent to the system console asking if the I/O descriptions on the save/restore medium are to be restored. QINRIO is called if the response is affirmative.
    - QINRIO–Restore I/O Configuration: This module deletes all existing I/O descriptions except for the system console and save/restore device(s). The save/restore component is called to restore the I/O descriptions from the save/restore medium.
      - QINRIOOH–Restore I/O Description Object Handler: This module, after initial processing by the save/restore component, receives the I/O description data and creates the appropriate logical unit description, control unit description, or network description.

QINFIXUP–Release Dependent Initialization: This module creates job and output queues and grants private authorities. In addition, this module performs other functions that are dependent upon the specific CPF release/modification level.

Installation Process Overview

Figure IN-1 and the following text describe the relationship of the installation component to other CPF components and functions. It also gives an overview of the installation process.

1 The AIPL machine interface source data consists of the initial user profile template, the program template for QINIT or QINITT, and the initial process definition template. This data resides on the save/restore medium as the first file. The AIPL procedure uses this source data to create an initial process and transfers control to the encapsulated form of QINIT or QINITT.

The purpose of QINIT and QINITT is to load the second file on the save/restore medium, QINSTALL. To perform this, a logical unit description for the diskette device is created. The QINSTALL module is then created and loaded into no context; control is transferred to it. Also, an ICO (installation communication object) is created. The ICO is used to save error messages and trace information for QINIT, QINITT, and QINSTALL, and contains data and pointers used by QINCPF.

If a termination error occurs, this object can be located and displayed to provide further debugging information along with any console messages. The existence of the ICO indicates to the start CPF process module that an installation is in progress and special CPF initialization must take place. This object is destroyed by QINCPF.

2 QINSTALL initially resolves to the QSYS context. If this context is not found or is damaged, a new one is created. The ICO is inserted into this context.

An logical unit description for the console is created, and some initialization of the associated spaces for both this logical unit description and the diskette logical unit description is performed.

QINSTALL then creates the following objects, if they do not already exist or are damaged:

- Required system user profiles
- Authorized users table that contains entries for the system user profiles
- Required system libraries (with the exception of the spooling library which is created in the start CPF process)

After the preceding objects are created, QINSTALL displays a prompt screen requesting the type of installation to be performed. The user may request the destruction of existing noninstalled CPF-created objects (cold start request) and/or request that no objects be loaded from the load/dump media. When objects are loaded from the media, this is called a normal install; otherwise, it is referred to as an abbreviated install (it should be apparent that abbreviated installs can only be performed on a system already containing CPF).

If the installation is from magnetic tape, a control unit description and a logical unit description for the tape controller and device are created. Initialization of the associated spaces is performed for both the control unit description and logical unit description.

If objects are to be loaded, QINSTALL loads or creates and loads the CPF objects on the fourth save/restore file and places them into the appropriate libraries. Each object is owned by the profile that owned it when the Save System command was executed.

The system-wide entry point table is created. Any previous entry point table is destroyed. This table is a space object containing resolved system pointers to those programs whose names are specified in the space object QLINMTBL.

Finally, QINSTALL creates the process definition template for the initial CPF process. This process definition template is stored into the machine attributes area and is used to both initiate CPF during AIPL and for subsequent IMPLs. The initial CPF process is initiated and, when QINSTALL receives a process-initiated event from the hardware, terminates itself.

3 When an installation is in progress, the ICO will exist on the system. As a result, the start CPF process performs some extra initialization. One of the modules called is QINCPF.

4 QINCPF calls common data management to send a display to the system console, requesting whether or not the operator wants to restore the I/O configuration file to that configuration saved with the system on the save/restore medium. If the response is yes, QINRIO is called (see 6). QINCPF then deletes certain system-supplied data base files and subsystem descriptions through an interface to the librarian component. It calls save/restore to restore data base files and subsystem descriptions. Following this, QINFIXUP is called (see 5). QINCPF then returns control to its caller.

5 QINFIXUP creates job and output queues. Private authorities are granted for certain system objects. For objects created by QINSTALL, information text is retrieved from the CPF message file and stored in the QSYS library. Additionally, this module performs other functions that are dependent on the specific CPF release/modification level.

6 QINRIO is called by QINCPF when the I/O descriptions are to be restored from the save/restore medium. The librarian component is called to provide a list of logical unit descriptions, control unit descriptions, and network descriptions. All descriptions except the system console and save/restore device(s) are deleted through an interface to the device configuration component. The save/restore component is then invoked to restore the I/O descriptions saved on the save/restore medium.

7 QINRIOOH is called for each I/O description being restored. It interfaces with the device configuration component to create the device description passed from save/restore. The security component is called to grant appropriate authorities and to transfer ownership of the descriptions as necessary. Save/restore is informed of the final disposition of the description, and control is returned to the caller.

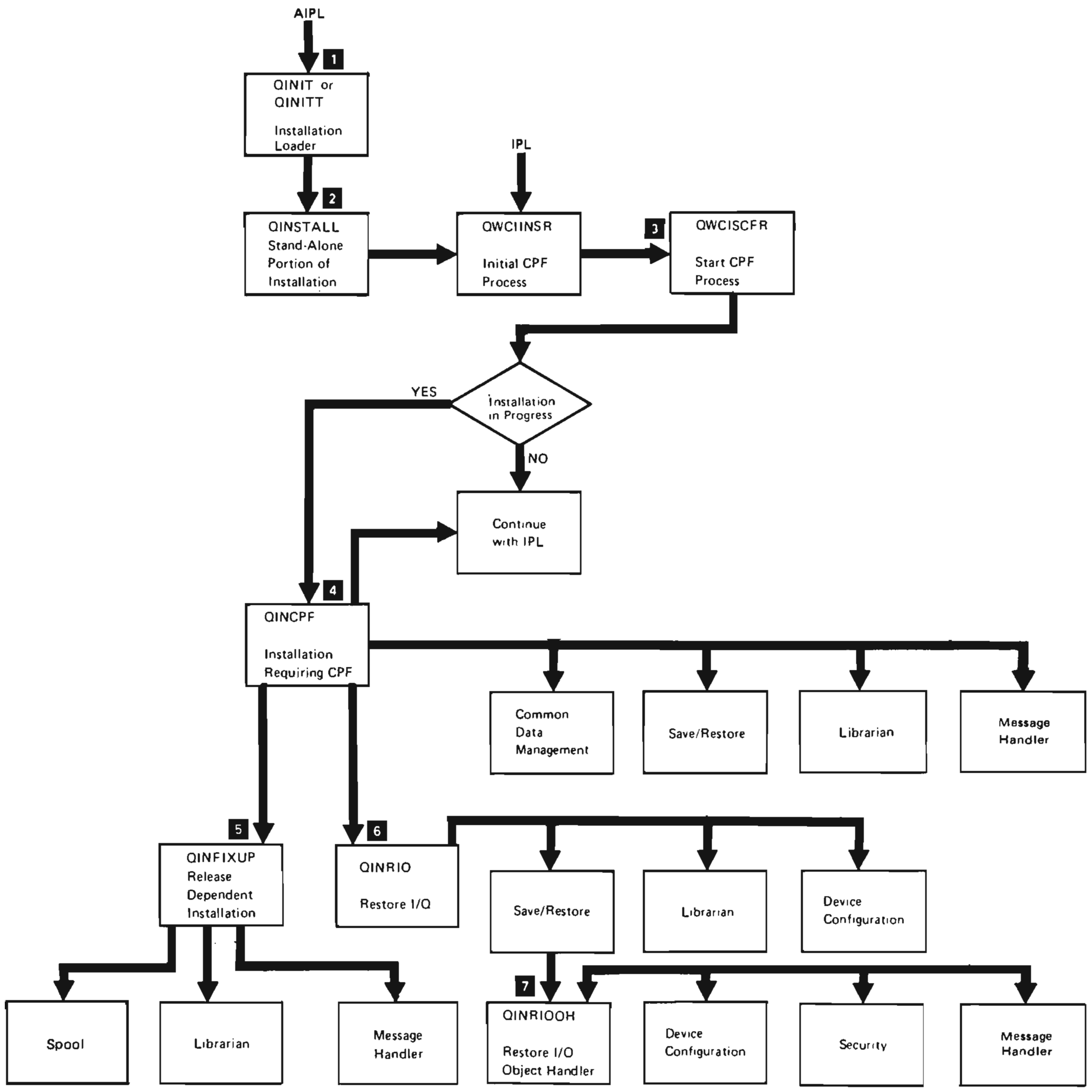


Figure IN-1. Installation Process Overview

Journal Management

INTRODUCTION

The journal management component of the CPF (control program facility) provides the user a means of recording changes made to data base files in an object.

The system creates a journal entry in a journal receiver when a change is made to a physical data base file, when a change is made to a journal object, or when the user requests that an entry be added. Only changes to physical files are recorded in a journal, regardless of how the journal operation is performed.

The user controls, via commands, the following functions:

- Command processing program: Executing the create, change, delete, or display of a journal, and the create, delete, or display of a journal receiver. Beginning and ending the journaling of a physical file. Reapplying journal entries, sending a user defined journal entry, retrieving a journal entry, comparing journaled images, and displaying the journal menu.

- Event/exception handling: Handling events or exceptions associated with journaling.

- Save/restore objects: Saving and restoring a journal or a journal receiver.

- Recovery: Recovering from an incomplete journal operation.

Entry to Journal Functions

The command processing program functions are visible to the users of journal management. The event/exception handling functions, save/restore object functions, and recovery functions appear automatic to the users.

The user entry to most of the journal functions is through the command analyzer. Figure JO-1 and the following text describe the paths of invocation of the journal management modules that provide the command processing program functions, event/exception handling functions, save/restore object functions, and recovery functions.

1 The command analyzer calls the command processing programs.

  A QJODLTJN and QJODLTRC are called by QLIDLOBJ.

2 The event/exception handling functions, save/restore object functions, and recovery functions are entered via a macro of the journal component or a call from the data base or save/restore component.

  B QDBRCIPS calls QJOJEJRC if a corresponding data base recovery object is found in QRECOVERY. If an entry for a journal receiver is found on the machine initialization status record the ?JORECVR macro may call QJOCDRJR, QJOCHJNR, or QJORRDIR. The ?JORECVR macro determines if a journal entry should be recovered by checking information stored in the journal control block and the receiver directory. If a create, delete, or restore of a journal was in progress, QJOCDRJR is called. If a change journal was in progress, QJOCHJNR is called. If an operation on the receiver directory was in progress, QJORRDIR is called.

  QDBRCIPS calls QJORTHRS if an entry for a journal receiver is found on the machine initialization status record that indicates the receiver threshold value was exceeded.

C CPF data base modules monitor for the vertical microcode entry-not-journaled exception. The handler defined is QJOXENNJ.

D QDBMOVFI, QDBRNMFI, QDBMVRFR, or QDBRNMME calls QJOCHGJD and QJOSNDJE.

E The CPF save/restore component calls save/restore object modules to handle the saving and restoring of a journal or a journal receiver.

F The event handling modules are called to handle events and send the messages to the appropriate message queue.

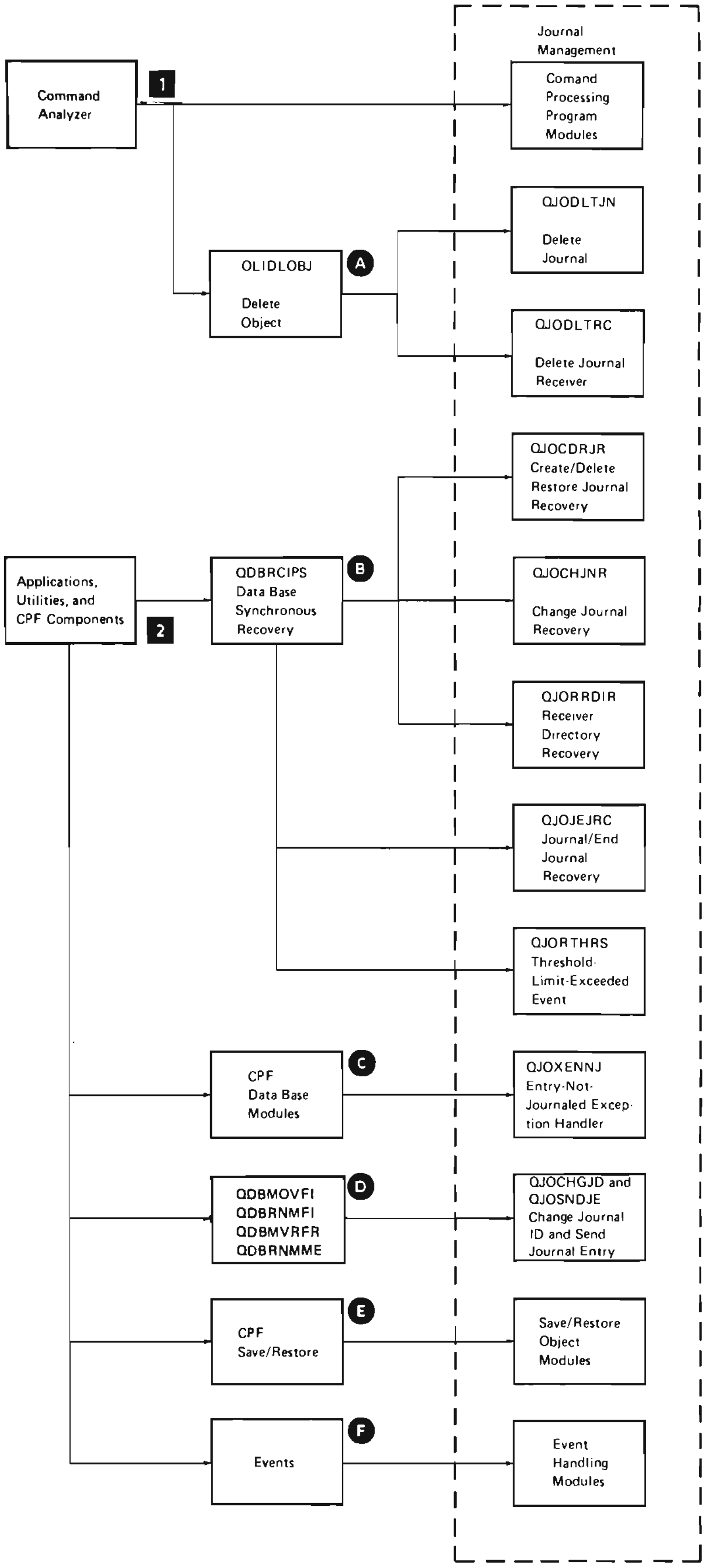

Journal Management
Command Analyzer
1
Comand Processing Program Modules
OLIDLOBJ Delete Object
A
QJODLTJN Delete Journal
QJODLTRC Delete Journal Receiver
QJOCDRJR Create/Delete Restore Journal Recovery
Applications, Utilities, and CPF Components
2
QDBRCIPS Data Base Synchronous Recovery
B
QJOCHJNR Change Journal Recovery
QJORRDIR Receiver Directory Recovery
QJOJEJRC Journal/End Journal Recovery
QJORTHRS Threshold-Limit-Exceeded Event
CPF Data Base Modules
C
QJOXENNJ Entry-Not-Journaled Exception Handler
QDBMOVFI QDBRNMFI QDBMVRFR QDBRNMME
D
QJOCHGJD and QJOSNDJE Change Journal ID and Send Journal Entry
CPF Save/Restore
E
Save/Restore Object Modules
Events
F
Event Handling Modules

Figure JO-1. Entry to Journal

GENERAL OVERVIEW

Journal Management Modules

The journal management component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry into the component. Indentation of a module shows its dependency on a previous module.

Command Processing Modules

-->QJOCRTJN–Create Journal (CRTJRN)[1]: This module creates a journal object and attaches journal receivers to it.

- QJOGENJD–Generate Journal ID: This module generates a journal ID and starts journaling.

-->QJODLTJN–Delete Journal (DLTJRN): This module detaches receivers, stops journaling on them, and deletes a journal.

-->QJOCRTRC–Create Journal Receiver (CRTJRNRCV)[1]: This module creates a journal receiver.

-->QJODLTRC–Delete Journal Receiver (DLTJRNRCV): This module deletes a journal receiver.

-->QJOCHGJN–Change Journal (CHGJRN)[1]: This module changes the operational and/or creational attributes of a journal. Changing the operational attributes involves detaching the currently attached journal receivers and attaching new journal receivers.

- QJOGENJD–Generate Journal ID: This module generates a journal ID and starts journaling.
- QJOCRTRC–Create Journal Receiver (CRTJRNRCV): This module creates a journal receiver.

-->QJODSPJA–Display Journal Attributes (DSPJRNA)[1]: This module displays the operational and creational attributes of a journal to the work station or to a spooled printer.

- QJODSPRC–Display Journal Receiver Attributes (DSPJRNRCVA): This module displays the attributes of a journal receiver.
- QJODLTRC–Delete Journal Receiver (DLTJRNRCV): This module deletes a journal receiver.

-->QJODSPJE–Display Journal (DSPJRN)[1]: This module displays the journal entries contained on one or more of the journal receivers to the work station, or a spooled printer output, and/or a data base output file.

- QJODJEHP–Display Journal Help Processor: This module displays the help screens for the Display Journal command.

-->QJOJNMNU–Display Journal Menu (DSPJRNMNU)[1]: This module displays the primary journal menu, and processes user requests.

- QJOJMNHP–Journal Help Processor: This module is called to display the help screen.
- QJOJSTAT–Journal Status: This module processes the request for journal status, and displays the status of a journal.
- QJOJNRCY–Journal Recovery: This module processes the request for recovery functions, and displays the journal recovery menu.
  - QJOJMNHP–Journal Help Processor: This module is called to display the help screen.
  - QJORYFIL–Recover File: This module processes the request for forward or back-out recovery from the journal recovery menu.
    - QJOJMNHP–Journal Help Processor: This module is called to display the help screen.
  - QJORYJRN–Recover Damaged Journal: This module processes the request for recovery of damaged journals from the journal recovery menu.
    - QJORYDIR–Recover Directory: This module reassociates all applicable receivers on the system with the recovered journal.

[1]This module is a CPP (command processing program).

QJORYRCV–Recover Journal Receiver: This module processes the request for recovery of damaged journal receivers from the journal recovery menu.

QJOJMNHP–Journal Help Processor: This module is called to display the help screen.

QJOJNLST–Journal Selection List: This module processes the request for the journal selection list from the primary journal menu.

QJOJNCMD–Journal Command: This module processes the request for the journal commands menu from the primary journal menu or journal recovery menu.

-->QJODSPRC–Display Journal Receiver Attributes (DSPJRNRCVA)[1]: This module displays the attributes of a journal receiver.

QJODSPRC–Display Journal Receiver Attributes (DSPJRNRCVA): This module displays the attributes of a journal receiver.

-->QJOJRNPF–Journal Physical File (JRNPF)[1]: This module begins journaling changes for a specific physical file to a journal.

QJOGENJD–Generate Journal ID: This module generates a journal ID and starts journaling.

-->QJOENDJN–End Journal (ENDJRNPF)[1]: This module ends journaling changes for a physical file.

-->QJORTVJE–Retrieve Journal Entry (RTVJRNE)[1]: This module retrieves a journal entry from a journal and places it into a set of user-defined CL variables.

QJODSPJE–Display Journal Entry: This module retrieves the requested journal entry from the journal, converts the entry from internal to external format, and returns the converted entry to QJORTVJE.

-->QJOSNDJE–Send Journal Entry (SNDJRNE)[1]: This module places a journal entry on a journal.

QJOGENJD–Generate Journal ID: This module updates the journal ID cross-reference table.

-->QJOREAPY–Reapply Journal Changes (APYJRNCHG/RMVJRNCHG)[1]: This module applies/removes journal changes to/from a specified file.

QJOREEXT–Reapply Invocation Exit: This module provides an invocation exit for module QJOREAPY.

-->QJOCMPJE–Compare Journal Images (CMPJRNIMG)[1]: This module compares images of record level changes recorded for a file.

The following module is invoked by the data base object handler for the Move Object (MOVOBJ), Rename Object (RNMOBJ), and Rename Member (RNMM) commands:

-->QJOCHGJD–Change Journal ID: This module updates object names in a journal ID cross-reference table that exists in the associated space of a journal receiver.

Event/Exception Handling Modules

-->QJONJEVT–Entry-Not-Journaled Event: This module handles the entry-not-journaled events and sends the messages to the system operator.

-->QJORTHRS–Threshold-Limit-Exceeded Event: This module handles the threshold-limit-exceeded events and sends the messages to the user specified message queue. This module is also invoked by QDBRCIPS during recovery.

-->QJORUEVT–Receiver-Unusable Event: This module handles the receiver-unusable events and sends the messages to the system operator.

-->QJOXENNJ–Entry-Not-Journaled Exception Handler: This module sends an inquiry to the system operator.

[1]This module is a CPP (command processing program).

Save/Restore Object Modules

-->QJORSTJN–Restore Journal: This module performs the object handler function necessary to restore a journal.

QJOCRTRC–Create Journal Receiver (CRTJRNRCV): This module creates a journal receiver.

QJOGENJD–Generate Journal ID: This module generates a journal ID and starts journaling.

-->QJORSTRC–Restore Journal Receiver: This module performs the object handler function necessary to restore a journal receiver.

QJOGENJD–Generate Journal ID: This module generates a journal ID and starts journaling.

-->QJOSAVJN–Save Journal: This module performs the object handler function necessary to save a journal.

-->QJOSAVRC–Save Journal Receiver: This module performs the object handler function necessary to save a journal receiver.

Recovery Modules

-->QJOCDRJR–Create/Delete/Restore Journal Recovery: This module performs the recovery of an interrupted create, delete, or restore journal operation.

-->QJOCHJNR–Change Journal Recovery: This module performs the recovery of an interrupted change journal operation.

QJOGENJD–Generate Journal ID: This module generates a journal ID and starts journaling.

-->QJOJEJRC–Journal/End Journal Recovery: This module performs journal recovery, including the backout or completion, of an interrupted journal physical file or end journal physcial file operation.

QJOGENJD–Generate Journal ID: This module generates a journal ID and starts journaling.

-->QJORRDIR–Receiver Directory Recovery: This module performs the recovery of an interrupted modification of the receiver directory.

Journal ID Generation

A journal ID is generated the first time an object is journaled. The journal ID associates journal entries to the appropriate object name. This association is maintained in a journal ID cross-reference table contained in the associated space of a journal receiver. A journal ID is assigned to an object for the object's life on the system; if journaling is ended and restarted for the object, the same journal ID will be used. The journal ID for an object is also preserved across a save/restore for the object.

A journal ID is generated by using a compressed version of the machine serial number, the segment identifier of the journal, and a 4-byte number that automatically increases by one. This generation scheme is used to provide journal ID to object uniqueness for each object journaled to a given journal, for each journal on a given system, and for each system.

Receiver Directory Management

A directory of all journal receivers associated with a journal is maintained in the associated space of the journal. This is called the receiver directory. When a journal receiver is associated with a journal through a Create Journal command, a Change Journal command, or a Restore Journal command, an entry for that journal receiver is added to the receiver directory. An entry will also be added to the receiver directory when a journal receiver is restored on the system and a directory entry does not yet exist for that journal receiver. If a journal receiver that is associated with a journal is deleted, its corresponding entry is removed from the appropriate receiver directory.

Each entry in the receiver directory contains a receiver identifier that is used to determine where each journal receiver belongs in the receiver chain. Multiple receiver chains are found in the receiver directory when one of the following conditions occur:

- A journal is restored.
- A journal receiver is restored that was previously saved while attached to the journal.
- A journal receiver is restored and its next receiver is not on the system.
- A journal receiver is restored, without storage freed, and its next receiver is restored, with storage freed.
- A journal receiver from another system is restored.
- A damaged journal receiver is deleted from the middle of a receiver chain.
- An unusable set of journal receivers is detected during journal operations.

Journal Object Locking

During journal operations, locks are obtained on the objects involved in the operation. This ensures the integrity of the object or the information presented concerning the object. Figure JO-2 shows the journal object locks, and the space location locks.

When multiple routing steps are performing operations that reference or modify the journal ID cross-reference table or the receiver directory table, the operations must be controlled so that there is no conflict in modification or reference. These operations are synchronized across routing steps by symbolic locking. Any operation that modifies one of the tables obtains an exclusive, no read, space location lock on the table. Any operation that references one of the tables obtains a shared, read only, space location lock on the table.

If two routing steps concurrently attempt an operation on the same table, one of the routing steps will obtain the lock. The other routing step will enter a lock wait, time-out loop until the first completes its operation. The second routing step will then obtain the lock so the desired operation can be performed.

Process Event Masking

In order to prevent undesired interruptions and cancellation during critical journal operations, the routing step is masked from interruption by an asynchronous system condition. The routing step is masked only for interruptions that could cause inconsistencies in journal object information. Masking the routing step is performed by QJOCRTJN, QJODLTJN, QJOCRTRC, QJODLTRC, QJOCHGJN, QJOJRNPF, QJOENDJN, QJORSTJN, QJOSAVRC, QJORSTRC, QJOCHGJD, QJOGENJD, and QJORYDIR. In addition, the event handlers QJORTHRS and QJORUEVT perform their operations with the process masked. The event handler QJONJEVT performs its operations with the process unmasked since it may process several events in one invocation.

When information is being gathered that requires a space pointer lock on the journal ID cross-reference table or the receiver directory table, QJOREAPY, QJODSPJA, and QJODSPJE prohibit cancel request. When a space pointer lock is not held, these operations are always interruptible and a cancel request will be allowed.

	Object Locks					Space Location Locks	
	Library	Journal	Journal Receiver	File	Member	Receiver Directory	Journal ID Cross-Reference
QJOCHGJD		LSUP				LSRO	LENR
QJOCHGJN		LENR	LENR[4]			LENR	
QJOCMPJE		LSRD	LSRD	LSRO	LSRD	LSRO	
QJOCRTJN		LENR	LENR				LENR
QJODLTJN	LSUP	LENR	LENR				
QJODLTRC	LSUP	LSRD[2]	LENR			LENR	
QJODSPJA		LSRD				LSRO	LSRO
QJODSPJE		LSRD	LSRD[9]			LSRO	LSRO
QJODSPRC			LSRD				
QJOENDJN		LEAR		LENR[3]			
QJOGENJD		LSUP	LENR			LSRO	LENR
QJOJNLST		LSRD					
QJOJNRCY		LENR					
QJOJRNPF		LSUP		LENR[3]			LENR
QJOJSTAT		LENR					
QJOREAPY		LSRO[6]	LSRD	LSRO[6]	LSRD[7]	LSRO	
QJORSTJN	LSUP	LENR	LENR				LENR
QJORSTRC	LSUP	LSUP[2]	LENR			LENR[2]	LENR
QJORUEVT						LENR	
QJORYDIR			LENR			LENR	
QJORYFIL				LSRO[8]			LSRO
QJORYJRN		LENR				LENR	LENR
QJORYRCV			LENR				
QJOSAVJN		LSRO					
QJOSAVRC		LSUP[2]	LSRO[1]			LENR[2]	
QJOSNDJE		LSUP		LSRO[5]			

[1]If save with storage freed is requested, LSRO will be LENR.
[2]Locks only acquired and receiver directory is updated if journal exists. The receiver directory will be locked LSRO during the reading of QJORSTRC.
[3]A series of data spaces or logical file control blocks may be locked LENR if logical files exist over this physical file.
[4]Both receivers to be attached and detached are locked.
[5]File is locked only if a specific file was requested on the command.
[6]Lock is changed to LSRD after parameter processing in QJOREAPY is completed.
[7]Member's data space is locked LENR to prevent a logical defined over the member from changing entries.
[8]Each file selected for recovery is locked LSRO. If a damaged file is selected, its dependent logicals are locked LENR.
[9]Each receiver in the receiver range is locked LSRD and the receivers in the chain from the receiver range up to and including the first receiver in the chain are also locked LSRD.

Figure JO-2. Journal Locking

INTRODUCTION

The Kanji component provides the function to perform ideographic conversion, manage ideographic dictionaries, and provide some of the functional support for the ideographic tables.

The first function that the Kanji component provides is the ideographic conversion interface. The ideographic conversion facility enables a user to enter ideographic data to an ideographic work station with either an alphameric or ideographic keyboard. The ideographic conversion function extends the data entry capability of an ideographic work station.

The second function that the Kanji component provides is the ideographic dictionary management functions. A user may create, display, edit, or delete an ideographic dictionary.

The third function that the Kanji component provides is the ideographic table support. A user may copy in or copy out an ideographic table from or to a diskette, check for an ideographic table, or delete an ideographic table. The editing capability of an ideographic table is supported through the EDTIGCTBL command function which is part of the character generation utility function.

GENERAL OVERVIEW

Kanji Modules

The Kanji component consists of the following modules:

-->QKJCPTBL–Copy Ideographic Table: This is an entry point module for the CPYIGCTBL command. This module allows a user to copy an ideographic table from or to a diskette or to copy an ideographic table from or to the system.

-->QKJCRDCT–Create Ideographic Dictionary: This is an entry point module for the CRTIGCDCT command. This module allows a user to create an ideographic dictionary.

-->QKJDSDCT–Display Ideographic Dictionary: This is an entry point module for the DSPIGCDCT command. This module displays the entries and their related words from the specified ideographic dictionary.

-->QKJEDIT–Edit Ideographic Dictionary: This is an edit module for the EDTIGCDCT command. This module allows a user to edit the related words of an entry in an IGC dictionary.

-->QKJEDSEL–Display Dictionary Entries: This is a select module of the EDTIGCDCT command. This module displays the dictionary entries that may be operated on by the user. The user can add entries to the dictionary, display entries in the dictionary, and remove entries from the dictionary.

-->QKJEROOT–Perform Normal Completion Operations: This is the ideographic conversion module. This module works in conjunction with the work station component (through the IGCCNV macro) to allow a user to perform ideographic conversion on an ideographic work station using either an alphameric or ideographic keyboard.

-->QKJHNTBL–Delete Ideographic Table: This is an entry point module for the DLTIGCTBL and the CHKIGCTBL commands. The DLTIGCTBL command will delete an ideographic table. The CHKIGCTBL command will check for the existence of an ideographic table.

-->QKJMDWRD–Add or Replace Dictionary Entries: This is the modify word module. This module will either add a new entry (and related words) or replace an existing entry (and related words) for an ideographic dictionary. This module is invoked through the MODWORD macro.

-->QKJRMENT–Remove Dictionary Entry: This is the remove entry module. This module will remove an entry (and its related words) from an ideographic dictionary. This module is invoked through the RMVENTRY macro.

-->QKJRTENT–Retrieve Dictionary Entry: This is the retrieve entry module. This module will retrieve the entries from an ideographic dictionary. This module is invoked through the RTVENTRY macro.

-->QKJRTWRD–Retrieve Dictionary Words: This is the retrieve word module. This module will retrieve the related words for an ideographic dictionary. This module is invoked through the RTVWORD macro.

-->QLIDLOBJ–Delete Dictionary from Library: This is an entry point module for the generic delete function. The DLTIGCDCT command will use this module as its command processing program. This module will delete an ideographic dictionary from a library. The librarian component owns and maintains this module.

Librarian

INTRODUCTION

The librarian component of CPF (control program facility) lets the user group objects into named sets called libraries. An object becomes a member of a library when the object is created. The object can be moved between libraries or renamed. It is always a member of a library until it is deleted.

The librarian component provides the following functions:

- Address resolution-related functions
  - Resolves the address of an object
  - Fast intermodule linkage
  - Library search list
- Object manipulation functions
  - Check an object
  - Delete an object
  - Move an object
  - Rename an object
  - Duplicate an object
  - Other generic functions
  - Clear a library
  - Delete a library
  - Create a library
  - Object information repository and its manipulation
- Display functions
  - Display the libraries on a library list
  - Display an object description
  - Display a library's contents

GENERAL OVERVIEW

Librarian Modules

The librarian component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QLICHLBL–Change Library List Entry (CHGSYSLIBL/ADDLIBLE/RMVLIBLE)[1]: This module adds a library to, or removes a library from, the system or user portion of the library list.

-->QLICHLIB–Change (replace) Library List (RPLLIBL)[1]: This module replaces the user portion of the library list that is associated with a job, with a list specified on the Replace Library List command.

-->QLICLLIB–Clear Library (CLRLIB/DLTLIB)[1]: This module lets users delete all of the objects in a library and the library if they have authority.

    QLIDLFIL–Delete Files: This module deletes files from a library.

-->QLICRDUP–Create Duplicate Object (CRTDUPOBJ)[1]: This module creates duplicate objects in a different library or it creates duplicate objects with different names in the same library as the original object.

-->QLICRLIB–Create Library (CRTLIB)[1]: This module is used to create user-defined libraries.

[1]This module is a CPP (command processing program).

-->QLIDOBJD–Display Object Description (DSPOBJD)[1]: This module is used to display the description of objects to the user.

QLIOUTFL–Create/Validate Outfile: This module creates an outfile and outfile member, or validates an existing outfile for any display commands in the librarian which support the OUTFILE and OUTMBR parameters.

-->QLIDSLIB–Display Library (DSPLIB)[1]: This module lets the user obtain a listing of all the objects in each of a list of libraries. The list contains the names of the objects as well as their basic attributes.

QLIHNCMD–Execute Display Options: This module gets all the objects selected from a librarian selection display and calls the module(s) necessary to process the option chosen for each selected object.

-->QLIDSPLL–Display Library List (DSPLIBL)[1]: This module processes the Display Library List command to display the library list.

-->QLILIST–List Object: This module provides the way to obtain information about the contents of libraries.

-->QLIMVOBJ–Move Object (MOVOBJ)[1]: This module moves an object from one library to another library.

QLIMOVE–Move Object Under Adopted User Profile: This module moves an object from one library to another library or removes an object from a library while running under QSYS.

-->QLIMVOIR–Move OIR Record: This module moves the OIR source records of an object from the OIR of the source library to the OIR of the target library.

-->QLIDLOIR–Delete OIR Record: This module deletes OIR records from the OIR of a library.

-->QLIMROIR–Modify/Retrieve OIR Record: This module adds, modifies, and retrieves object information to or from an OIR.

-->QLIRNOBJ–Rename Object (RNMOBJ)[1]: This module is used to change the name of an object.

-->QLIRNOIR–Rename OIR Entries: This module renames the OIR entries for an object.

-->QLICNV–Convert Object Type: This module converts symbolic object type to machine interface type/subtype codes and converts machine interface type/subtype codes to symbolic object types.

-->QLIINSRT–Insert Object in Library: This module inserts an object and its OIR data into a library.

-->QLICLNUP–Library Clean-Up Routine: This module cleans up incompletely created or damaged libraries caused by system malfunctions that occur during execution of create library function.

-->QLIDLOBJ–Delete CPF Object Module: This module processes all LU delete commands to delete CPF objects from the system.

QLIFITYP–File Type Identification: This module determines the file type of each file (device, physical, logical, or derived).

-->QLICKOBJ–Check Object (CHKOBJ)[1]: This module checks the existence of an object, and optionally, checks the authorization the user has for the object.

-->QLIADOPT–Set Space Authority in System Pointer Under Adopted User Profile: This module sets space authority in a system pointer while running under QSYS.

-->QLIRCLIB–Reclaim Library: This module cleans up damaged libraries found during execution of the RCLSTG command.

QLICNVD–Convert Library OIR: This module converts the OIR of a library to the current release/modification level during the RCLSTG command. The library requiring conversion will previously have been a floating library.

-->QLIVLOIR–Validate OIR: This module removes OIR entries for which no object exists in the library.

[1]This module is a CPP (command processing program).

Object

An object is a named, separately addressable unit that has a set of attributes, a value, and a unique set of operational characteristics. The attributes associated with an object describe that object. The value of an object is the contents of the object itself. The operational characteristics of an object define how an object can be used and what operations can be performed against it.

Library

A library is a special type of object that is used to group other objects into named sets. The purpose of libraries is to provide a directory for the objects that the user creates. All objects, other than libraries, are required to reside in libraries. An object can only reside in one library at a time. Duplicate named objects cannot reside in the same library unless they have different object types. Object type is an additional qualification on an object.

Library List

A library list is an ordered list of libraries that is associated with each job during its execution. The library list determines which libraries are searched, and controls the order in which they are searched to resolve a reference to an object.

The library list is made up of two parts: a system part and a user part. The system part of the library list always precedes the user part, so it is always searched first. The user part of the library list can be changed by using the Replace Library List (RPLLIBL) command, the Add Library List Entry (ADDLIBLE) command, the Remove Library List Entry (RMVLIBLE) command, or the Change System Library List (CHGSYSLIBL) command. It can also be displayed using the Display Library List (DSPLIBL) command. Whenever the library list is changed, the change made in the list remains in effect for the duration of the job or until another Replace Library List command, Add Library List Entry command, Remove Library List Entry command or Change System Library List command is processed.

Check an Object

The existence of an object, and optionally, the authorization a user holds for the object can be checked using the Check Object (CHKOBJ) command.

Delete an Object

An object can be deleted from a library by one of the delete object commands. The librarian supports the DELETE generic function definition table.

Move an Object

An object can be moved from one library to another using the Move Object (MOVOBJ) command. Control unit descriptions, device descriptions, line descriptions, edit descriptions, user profiles, and libraries cannot be moved.

Rename an Object

The name of an existing object can be changed by the Rename Object (RNMOBJ) command. However, the IBM-supplied library (QSYS), job temporary library (QTEMP), control unit descriptions, device descriptions, line descriptions, and user profiles cannot be renamed.

Create Duplicate Object

An existing object can be duplicated by the Create Duplicate Object (CRTDUPOBJ) command; however, unit descriptions, device descriptions, edit descriptions, journals, journal receivers, libraries, line descriptions, and user profiles cannot be duplicated.

Other Generic Functions

The librarian component supports internal tables defining operation applicability, object structures, and routines performing the operations for other generic functions, such as grant, revoke, allocate, deallocate, save, restore, dump, suspend, display object locks, and change object ownership.

Clear a Library

The Clear Library (CLRLIB) command is used to delete objects from a library. The Clear Library command only deletes objects for which the user has object existence authority.

Delete a Library

The Delete Library (DLTLIB) command is used to delete a library. The library to be deleted is deleted only if all objects in it can be deleted.

Create a Library

The user can create as many libraries as desired and use them to hold other objects. After a library has been created, other objects can be created or moved into it.

Object Information Repository and its Manipulations

Some of the attributes of an object are not kept with the object. There are six types of information in this category:

- Text information
- Service information
- Save/restore information
- Special attribute information
- File reference function information
- Journal information

The information is stored in an OIR (object information repository). Every library has an OIR associated with it. The librarian provides ways to add, delete, modify, and retrieve information in the OIR. Function is also provided to rebuild an OIR that is damaged.

Display a Library List

The Display Library List (DSPLIBL) command displays the name, attribute, and text of each library on the library list. This display also distinguishes the system and user portions of the library list from each other.

Display an Object Description

This function displays the attribute information associated with an object or a set of objects. A basic, full, or service description of an object can be requested. The Display Object Description (DSPOBJD) command displays descriptions for all objects of a specified type, for all objects with a given generic name, or for objects with a given generic name and given type.

Display the Contents of a Library

This function displays all the objects contained in each of a list of libraries. The list of libraries can be specific libraries or the libraries contained in the user's library list or all the libraries the user is authorized to use, including those libraries publicly authorized.

Librarian Relationships to Other CPF Components

Figure LI-1 shows the relationship between the librarian component and other CPF components. The following components use the librarian for the indicated purposes:

1 Security uses the GRANT/REVOKE/TRANSFER generic function definition tables to control the operation of its grant object authority, revoke object authority, and change object ownership CPPs (command processing programs).

2 Save/restore uses the SAVE/RESTORE generic function definition tables to control the save and restore functions. It uses QLILIST to obtain objects from a library, and it uses OIR (object information repository) manipulation functions to save/restore OIR information along with objects.

3 Service uses the DUMP generic function definition table to perform the dump system object function. It also uses QLILIST to obtain information about objects in a library.

4 Work management uses the ALLOCATE/DEALLOCATE generic function definition table to perform allocate and deallocate functions. Work management also uses QLIDSPLL to display the library list of the job displayed by the Display Job (DSPJOB) command. It uses the create library function to create the QTEMP library. It uses QLICLNUP to check for and recover from damage to libraries during an IMPL.

5 All CPF components use the macros provided by the librarian to resolve an object address, to pass control to a module, and to convert symbolic object types to their internal representations.

6 Installation uses WWLICNVO to convert a library OIR if it is in an old format. It uses WWLIVOIR to check for and recover from damage to libraries QSYS, QSPL, QSRV, QGPL, and QRECOVERY. It also uses OIR manipulation functions to restore OIR information for objects that are installed.

Reclaim uses QLIRCLIB to check for and recover from damage to all libraries. It uses QLICNVO to convert any library OIR that is in an old format. It uses QLIVLOIR to delete old OIR entries that do not have an object in the library.

7 Librarian commands use the command analyzer to pass command parameter to the librarian CPPs.

8 Librarian uses work management to gain access to library QTEMP and to job structure control blocks, such as WCB, WCBT, and so forth.

9 Librarian uses security to verify, grant, and revoke authorizations.

10 Librarian uses the message handler to signal exceptions, send messages, move messages, and to retrieve exception data.

11 Librarian uses data management functions to perform I/O operations to support the display commands in the librarian component.

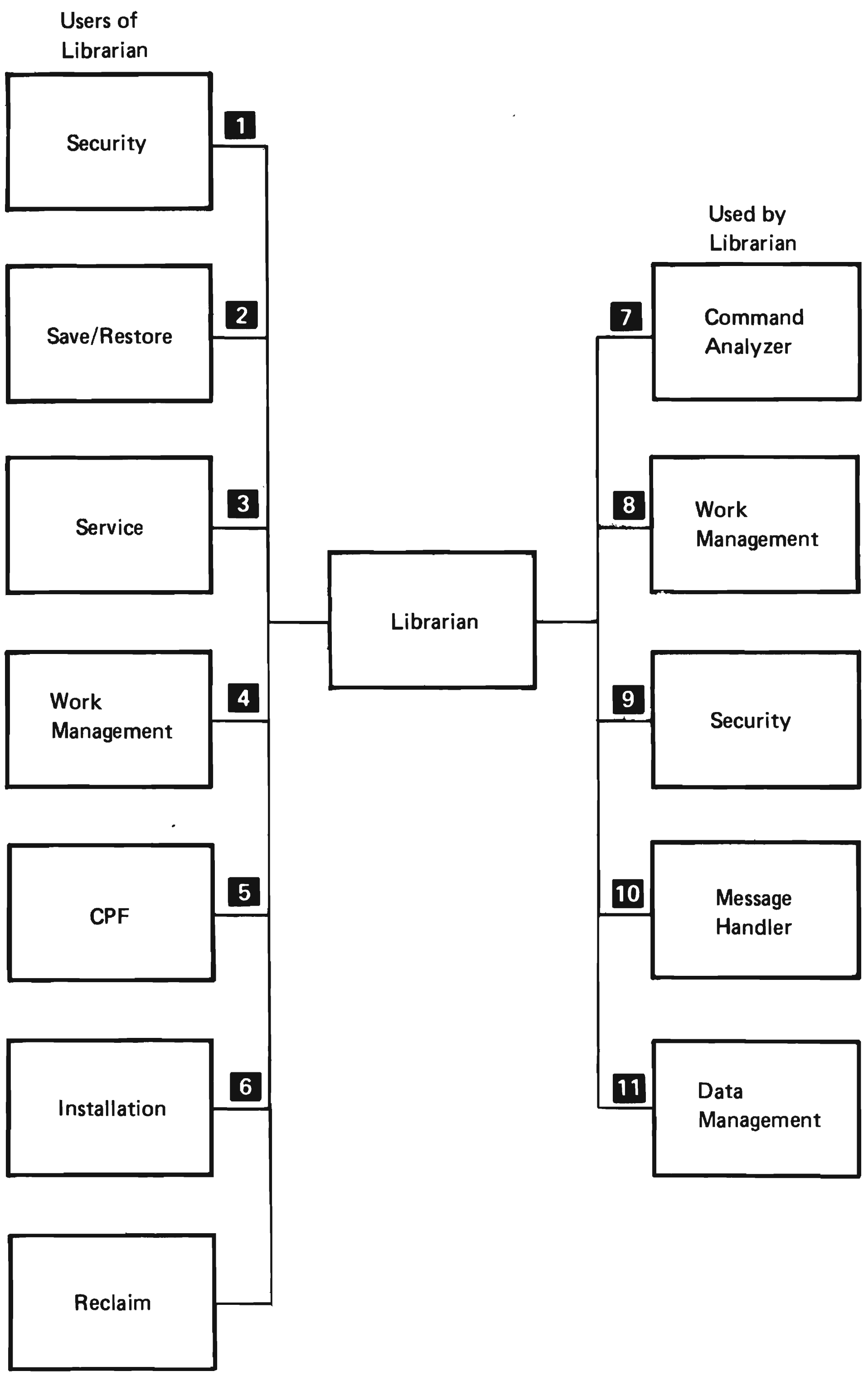


Figure LI-1. Relationship of Librarian to Other CPF Components

Replace Library List Command

Figure LI-2 and the following text describe the operation of the Replace Library List (RPLLIBL) command.

1 The command analyzer decodes a Replace Library List command and control is transferred to QLICHLIB.

2 QLICHLIB verifies that the library names specified by the user are unique and that the user is authorized to access these libraries.

3 QLICHLIB replaces the user portion of library list with the new list, if the libraries are unique, exist, are not damaged, and if the user is authorized to access them. Otherwise, the library list remains unchanged.

4 Control is returned to the caller.

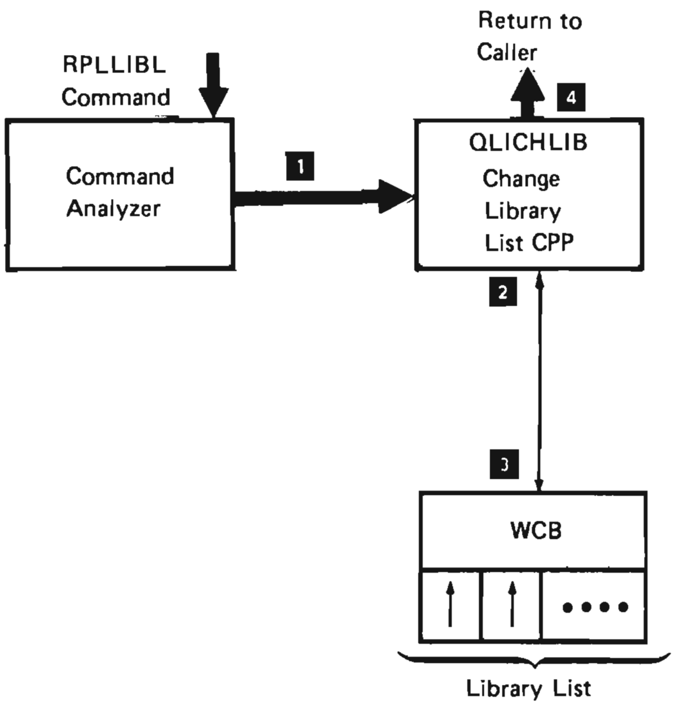


Figure LI-2. Replace Library List Command Overview

Clear Library Command

Figure LI-3 and the following text describe the operation of the Clear Library (CLRLIB) command.

1 The command analyzer decodes a Clear Library command and control is transferred to QLICLLIB.

2 QLICLLIB obtains a list of objects from the library.

3 If there are files in the library, QLICLLIB calls QLIDLFIL to delete the files.

4 For other objects in the library, QLICLLIB calls QLIDLOBJ to delete the objects.

5 Delete MPCI (master programming change index), if any.

6 Control is returned to the caller.

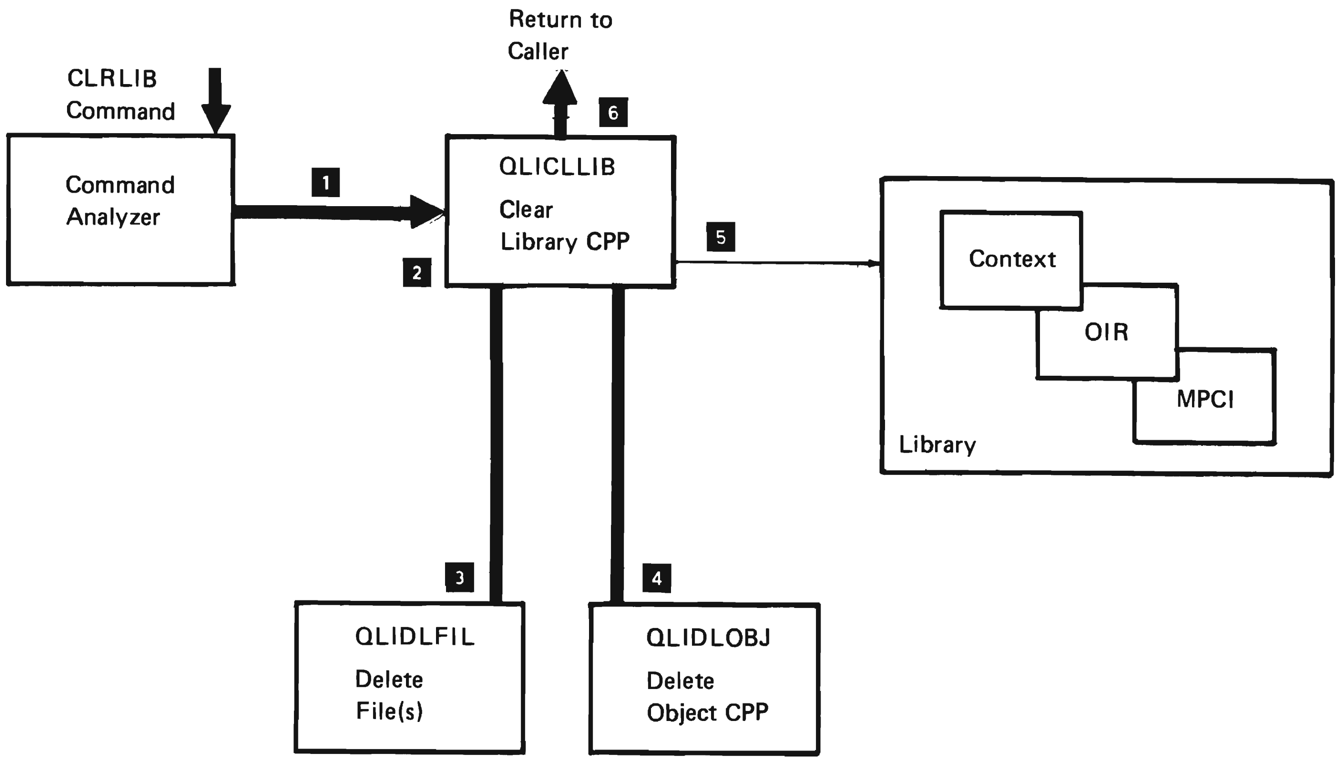


Figure LI-3. Clear Library Command Overview

Create Library Command

Figure LI-4 and the following text describe the operation of the Create Library (CRTLIB) command.

1 The command analyzer decodes a Create Library command and control is transferred to QLICRLIB.

2 QLICRLIB creates a library recovery object in QRECOVERY.

3 The context and the OIR of a library are created.

4 Text, special attribute, and service records are added to the OIR of the library.

5 Grant public authorization to the library. Grant authority for the library to the process group profile.

6 QLICRLIB deletes the library recovery object from QRECOVERY.

7 Control is returned to the caller.

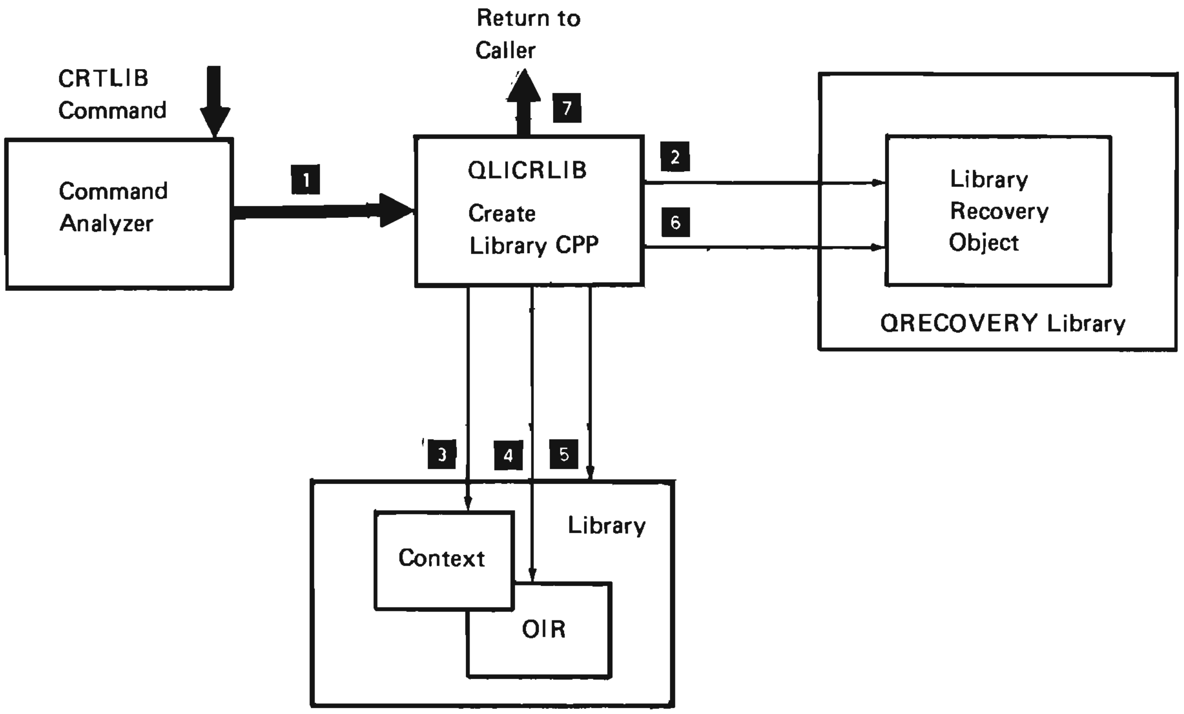


Figure LI-4. Create Library Command Overview

This page is intentionally left blank.

Delete Object Commands

Figure LI-5 and the following text describe the operation of the following commands:

Delete Class (DLTCLS)
Delete Command (DLTCMD)
Delete Control Unit Description (DLTCUD)
Delete Device Description (DLTDEVD)
Delete Data Area (DLTDTAARA)
Delete Edit Description (DLTEDTD)
Delete File (DLTF)
Delete Forms Control Table (DLTFCT)
Delete Job Description (DLTJOBD)
Delete Job Queue (DLTJOBQ)
Delete Journal (DLTJRN)
Delete Journal Receiver (DLTJRNRCV)
Delete Line Description (DLTLIND)
Delete Message File (DLTMSGF)
Delete Message Queue (DLTMSGQ)
Delete Output Queue (DLTOUTQ)
Delete Program (DLTPGM)
Delete Print Image (DLTPRTIMG)
Delete Subsystem Description (DLTSBSD)
Delete Session Description (DLTSSND)
Delete Table (DLTTBL)

1 QLIDLOBJ obtains control from the command analyzer.

2 QLIDLOBJ compares the type of the object against the DELETE generic function definition table to determine whether to invoke a program to perform the delete function.

3 QLIDLOBJ calls QDCDLCD to delete a control unit description.

4 QLIDLOBJ calls QDCDLLUD to delete a device description.

5 QLIDLOBJ calls QDCDLND to delete a line description.

6 QLIDLOBJ calls QSPHNSPQ to delete a job queue or an output queue.

7 QLIDLOBJ calls QMHDLMSQ to delete a message queue.

8 If multiple files are being deleted, QLIDLOBJ calls QLIFITYP to determine the file type (device, physical, logical, or derived).

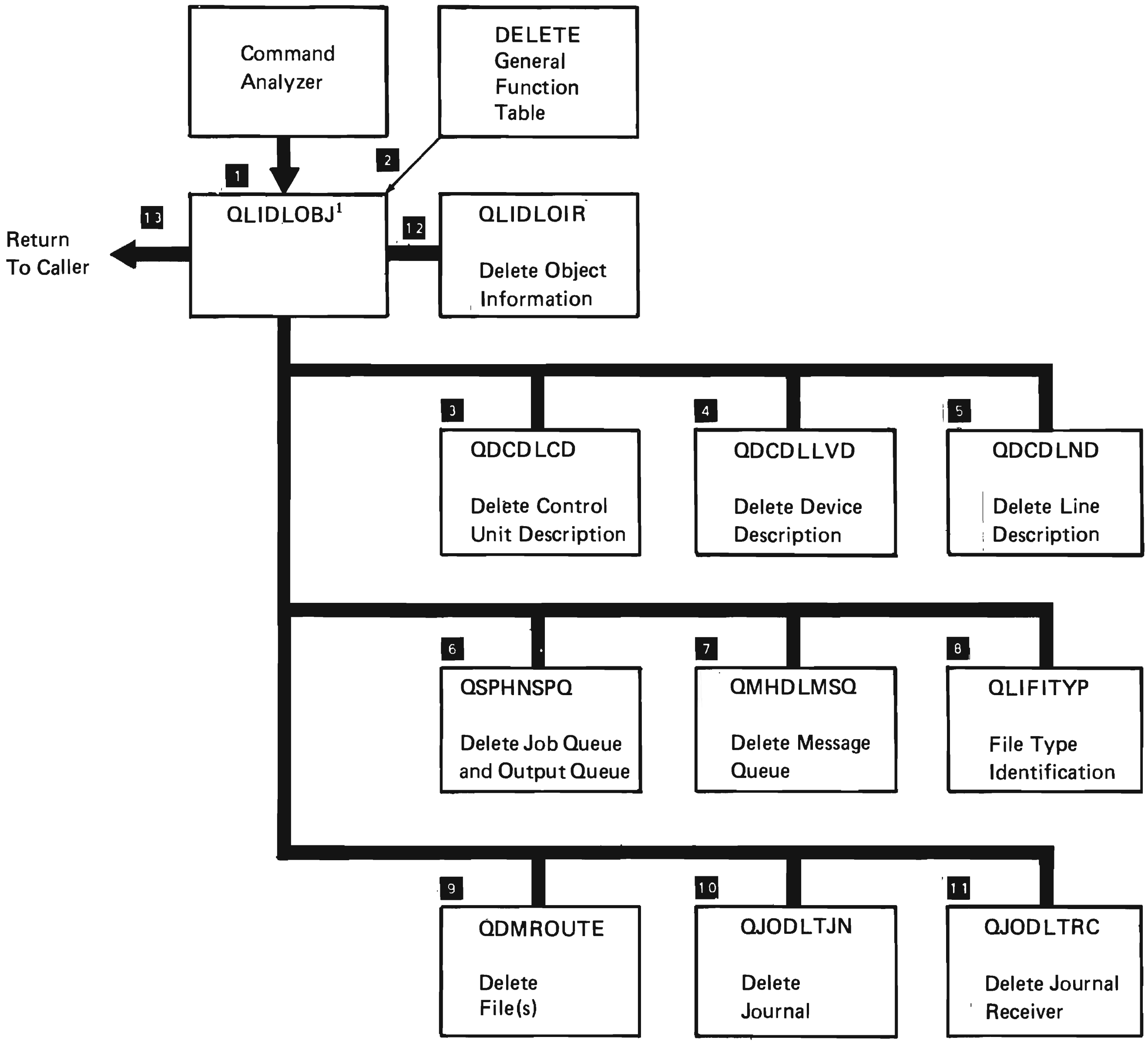
9 QLIDLOBJ calls QDMROUTE to delete a file.

10 QLIDLOBJ calls QJODLTJN to delete a journal.

11 QLIDLOBJ calls QJODLTRC to delete a journal receiver.

12 QLIDLOBJ deletes the object and its associated OIR records if the object type is CLS, CMD, JOBD, DTAARA, EDTD, FCT, MSGF, PGM, PRTIMG, SBSD, SSND, CHTFMT, SPADCT, GSS, or TBL.

13 Control is returned to the caller.


Command Analyzer
DELETE General Function Table
1
2
QLIDLOBJ[1]
13
Return To Caller
12
QLIDLOIR Delete Object Information
3
QDCDLCD Delete Control Unit Description
4
QDCDLLVD Delete Device Description
5
QDCDLND Delete Line Description
6
QSPHNSPQ Delete Job Queue and Output Queue
7
QMHDLMSQ Delete Message Queue
8
QLIFITYP File Type Identification
9
QDMROUTE Delete File(s)
10
QJODLTJN Delete Journal
11
QJODLTRC Delete Journal Receiver

[1] Delete class, command, data area, edit description, forms control table, job description, message file, program, print image, session description, subsystem description, chart format, spelling aid dictionary, or graphic symbol set.

Figure LI-5. Delete Object Commands Overview

Delete Library Commands

Figure LI-6 and the following text describe the operation of the Delete Library (DLTLIB) command.

1 The command analyzer decodes a Delete Library command and control is transferred to QLICLLIB.

2 QLICLLIB verifies that the library to be deleted is not one of the libraries in any job library list and the user has existence authority over the library.

3 QLICLLIB calls QLIDLOBJ to delete objects other than object type FILE from the library being deleted.

4 QLICLLIB calls QLIDLFIL to delete object type FILE objects.

5 If the library is cleared, the OIR and the context are destroyed. Otherwise, the library is not deleted.

6 Control is returned to the caller.

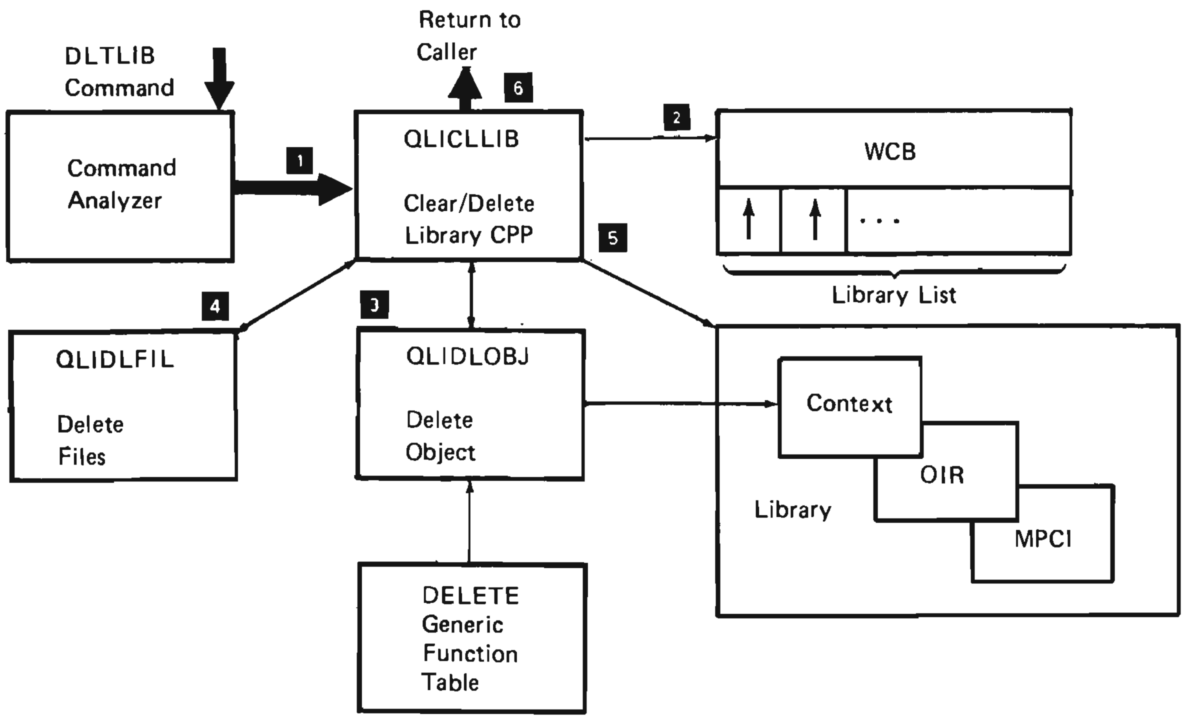


Figure LI-6. Delete Library Commands Overview

Display Library Command

Figure LI-7 and the following text describe the operation of the Display Library (DSPLIB) command.

1 The command analyzer decodes a Display Library command and control is transferred to QLIDSLIB.

2 QLILIST is called to retrieve basic information for the objects in a library when the output is to printer, or for an object selection display of a library's contents.

3 The output is formatted and then printed or displayed. Displayed output can be a library selection display or an object selection display of a library's contents. Printed output can be basic information for all objects in all specified libraries.

4 QLIHNCMD is called to process records selected from a library or object selection display. If the records selected are from a library selection display, QLIHNCMD calls QLIDSLIB to display an object selection display for each library.

5 QLIDOBJD is called to display *FULL or *SERVICE information about each object selected from an object selection display.

6 Control is returned to the caller.

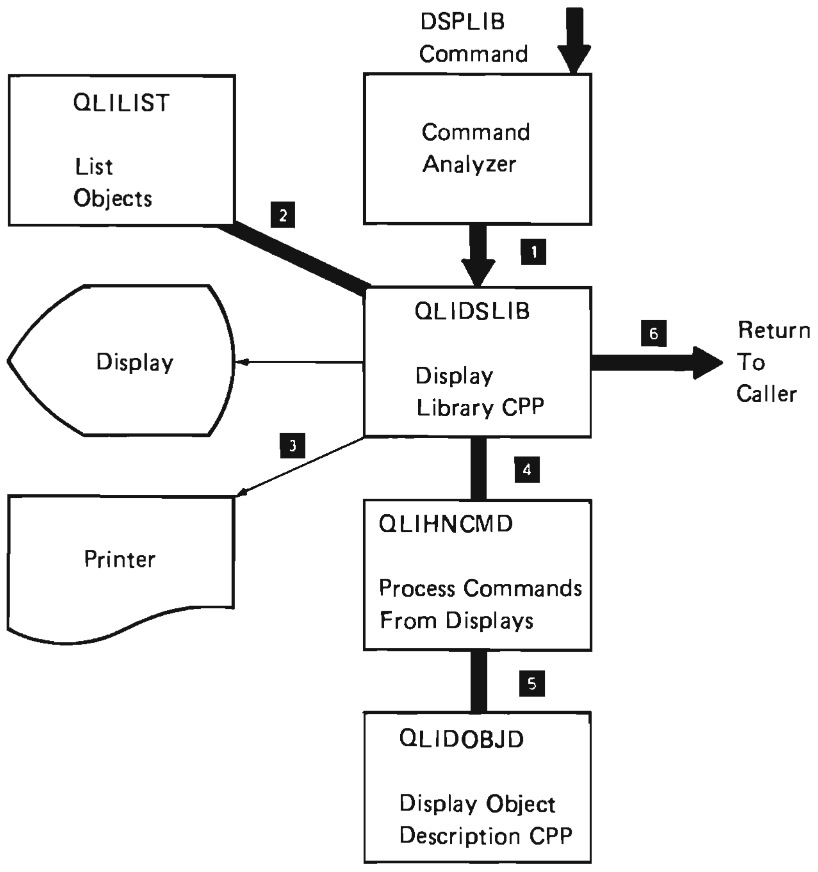


Figure LI-7. Display Library Command Overview

Display Object Description Command

Figure LI-8 and the following text describe the operation of the Display Object Description (DSPOBJD) command.

1 The command analyzer decodes a Display Object Description command and control is transferred to QLIDOBJD.

2 Invoke QLILIST to obtain descriptions for the specified objects.

3 Information about the object or objects is formatted and displayed or printed.

4 QLIHNCMD is called to process records selected from an object selection display. QLIHNCMD calls QLIDOBJD to display *FULL or *SERVICE information about each selected object.

5 OUTFILE and OUTMBR are created/validated if specified on command.

6 QLIOUTFL creates an empty output file.

7 QLIDOBJD puts data in the output file.

8 Control is returned to the caller.

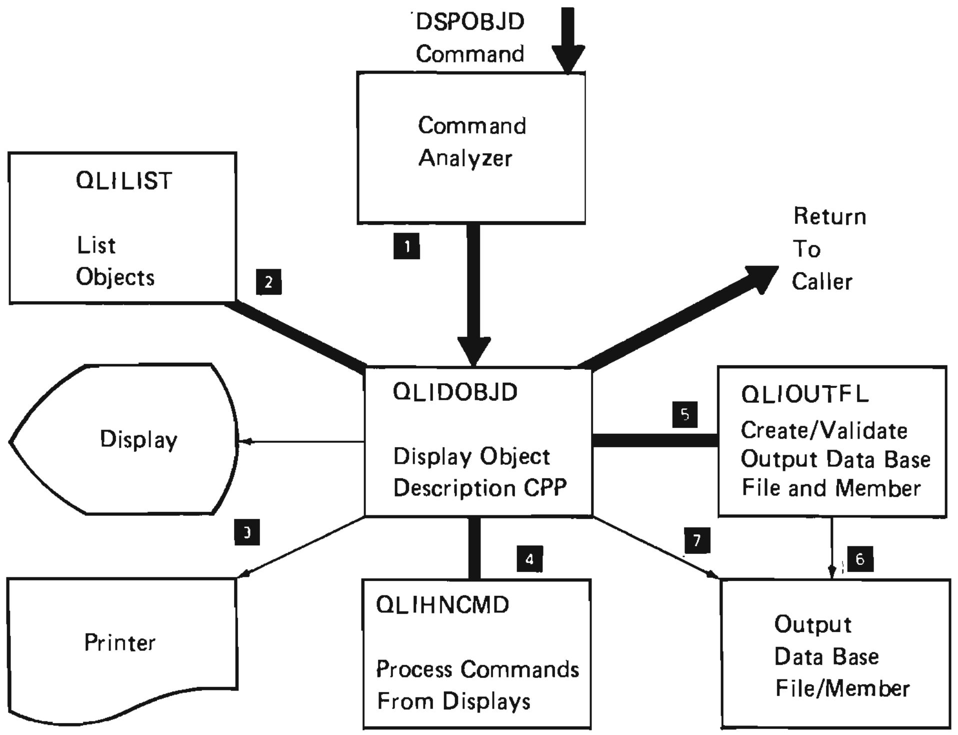

DSPOBJD Command
Command Analyzer
QLILIST
List Objects
Return To Caller
1
2
QLIDOBJD
Display Object Description CPP
Display
QLIOUTFL
Create/Validate Output Data Base File and Member
5
3
4
7
6
Printer
QLIHNCMD
Process Commands From Displays
Output Data Base File/Member

Figure LI-8. Display Object Description Command Overview

List Objects

Figure LI-9 and the following text describe the operation of the list objects function.

1 QLILIST is invoked by the ?LSTOBJ macro.

2 The requested information is retrieved from the library(s) or object(s).

3 A system pointer is returned to the caller of the macro pointing to the requested information.

4 Control is returned to the caller.

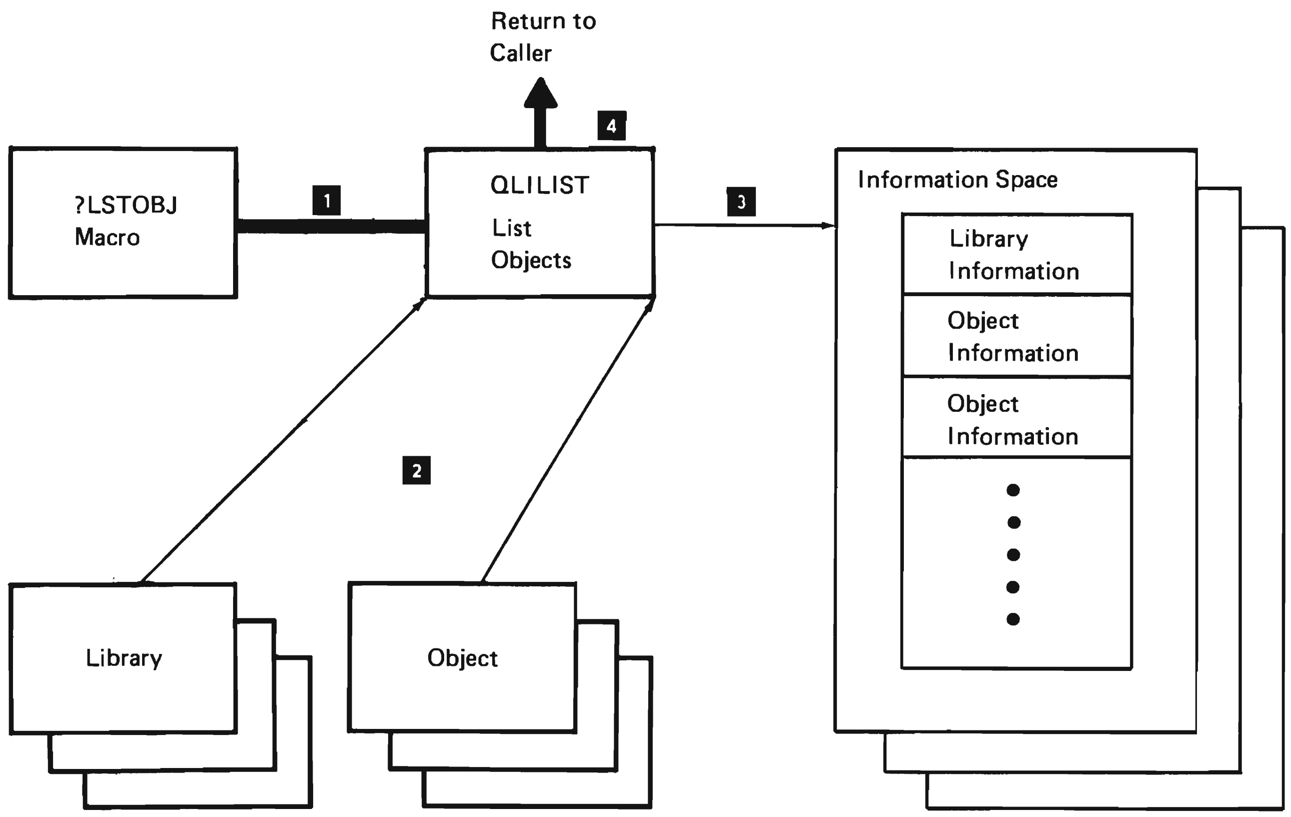


Figure LI-9. List Objects Overview

Move Object Command

Figure LI-10 and the following text describe the operation of the Move Object (MOVOBJ) command:

1 The command analyzer decodes a Move Object command and control is transferred to QLIMVOBJ.

2 The move generic function definition table is checked to determine function applicability, object structure, and the routine to be called to perform the function.

3 QLIMVOBJ calls QDMROUTE to move a file.

4 QLIMVOBJ calls QSPHNSPQ to move a job queue or an output queue.

5 QLIMVOBJ calls QMHMRCHK to move a message queue.

6 If the object is a standard object and does not need special processing, the object and its OIR records are moved to the target library.

7 QLIMVOBJ calls QLIMOVE to back out any move not completed.

8 Control is returned to the caller.

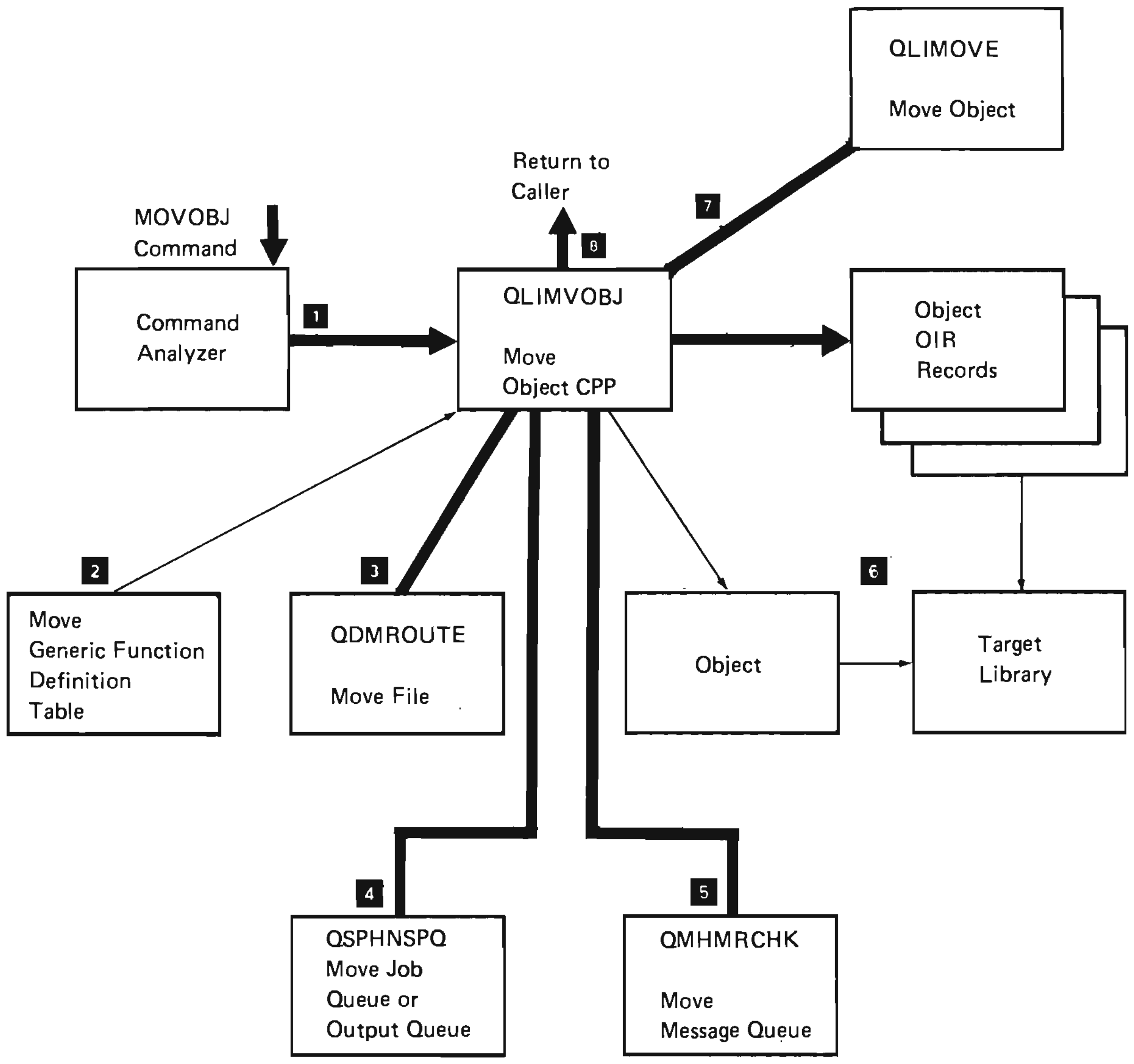


Figure LI-10. Move Object Command Overview

Rename Object Command

Figure LI-11 and the following text describe the operation of the Rename Object (RNMOBJ) command.

1. The command analyzer decodes a Rename Object command and control is transferred to QLIRNOBJ.

2. The rename generic function definition table is checked to determine function applicability, object structure, and the routine to be called to perform the function.

3. QLIRNOBJ calls QDMROUTE to rename a file.

4. QLIRNOBJ calls QMHMRCHK to rename a message queue.

5. QLIRNOBJ calls QSPHNSPQ to rename a job queue or an output queue.

6. If the object is a standard object and does not need special processing, the identification of the object and its OIR (object information repository) records are renamed as requested.

7. Control is returned to the caller.

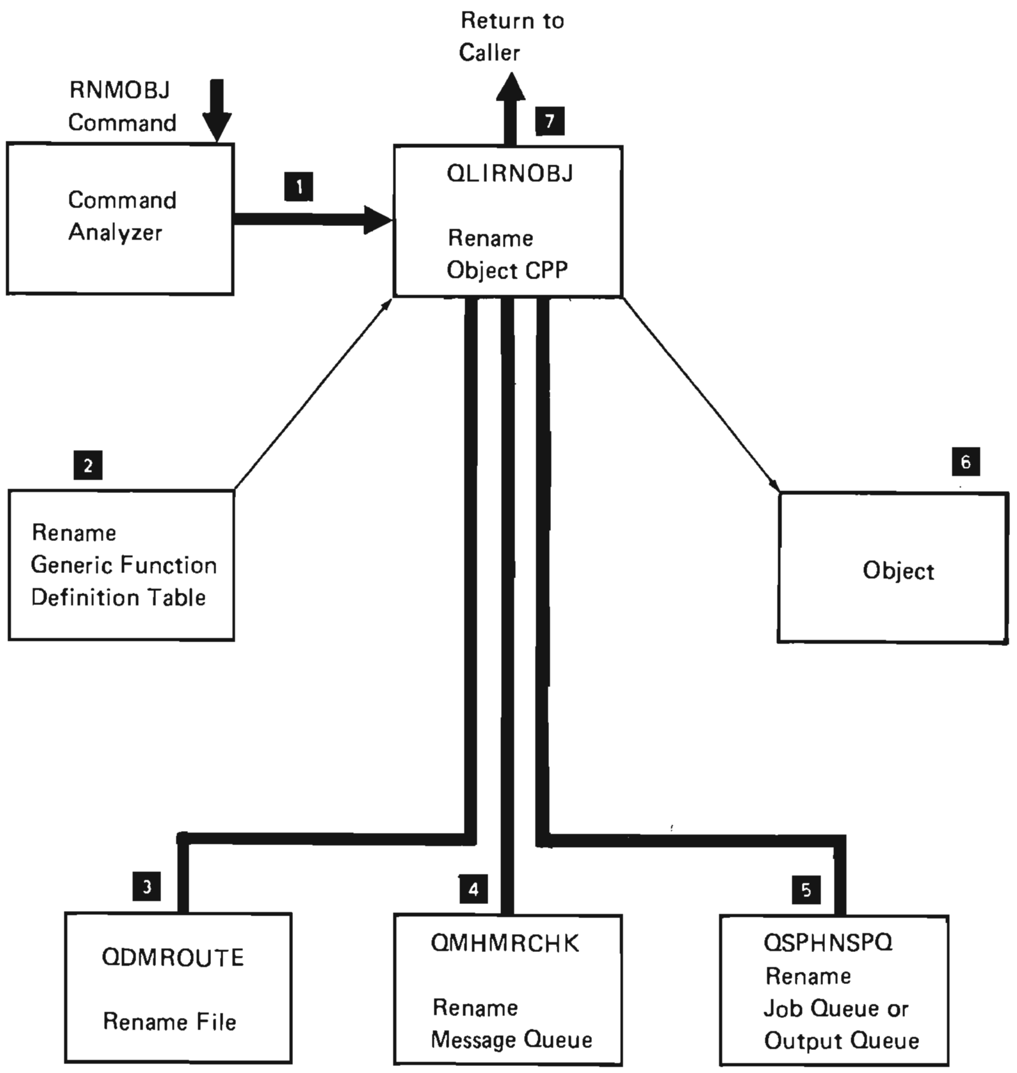


Figure LI-11. Rename Object Command Overview

Library Clean-Up During IPL

Figure LI-12 and the following text describe the operation of the library clean-up function during IMPL.

1 Control is passed to QLICLNUP from QWCISCFR during IPL (initial program load).

2 QLICLNUP examines QRECOVERY to determine if there are library recovery objects in it. The presence of a library recovery object indicates that a library was not completely created.

3 QLICLNUP deletes those libraries that are not completely created.

4 QLICLNUP removes library recovery objects from QRECOVERY.

5 QLICLNUP checks each library and pieces of that library on the system for marked damage and recovers the damaged pieces where possible.

6 QLICLNUP sets on the immediate update flag in the OIR index of each library on the system.

7 Control is returned to the caller.

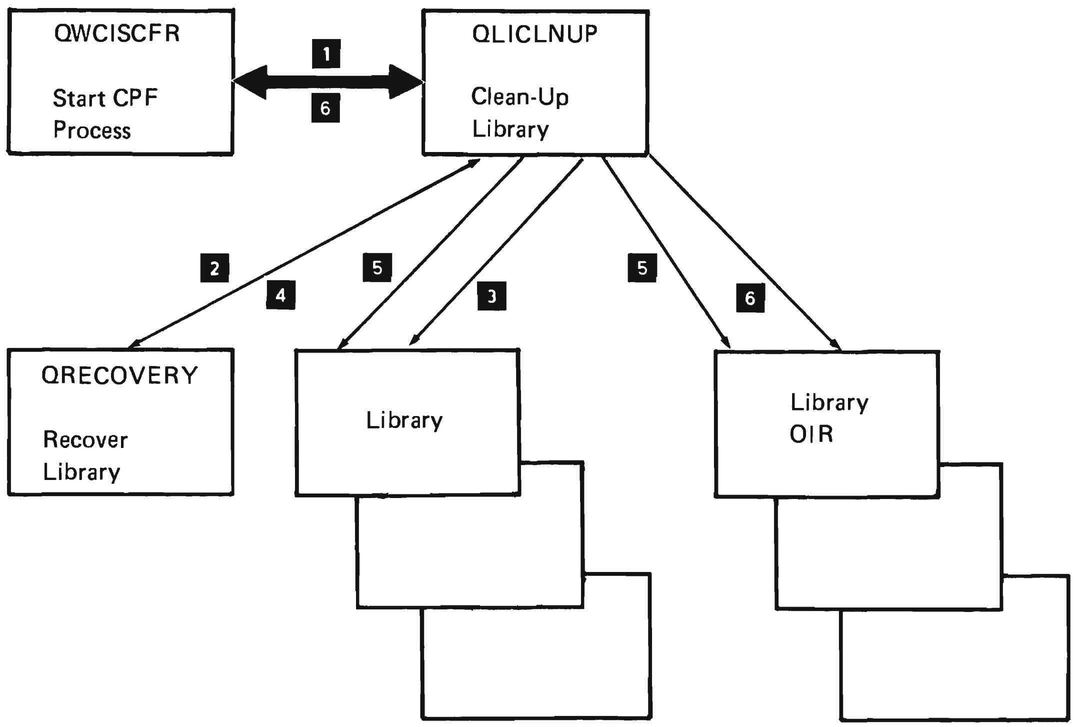


Figure LI-12. Library Clean-Up During IPL

Check Object Command

Figure LI-13 and the following text describe the operation of the Check Object (CHKOBJ) command.

1 The command analyzer decodes a Check Object command and passes control to QLICKOBJ.

2 QLICKOBJ resolves both the address of the addressing library of the object whose existence is to be verified, and the address of the object itself, to verify that the object does exist in the library.

3 QLICKOBJ verifies the authorization the user has to the object (if authorization verification is requested and the object is not a data base file or a member).

4 QLICKOBJ calls QLIADOPT to set space authority (*SPCAUT) in the system pointer to an object.

5 QLICKOBJ calls QDBCHKFI if the member is to be checked or authorization verification against a data base file is requested.

6 Control is returned to the caller.

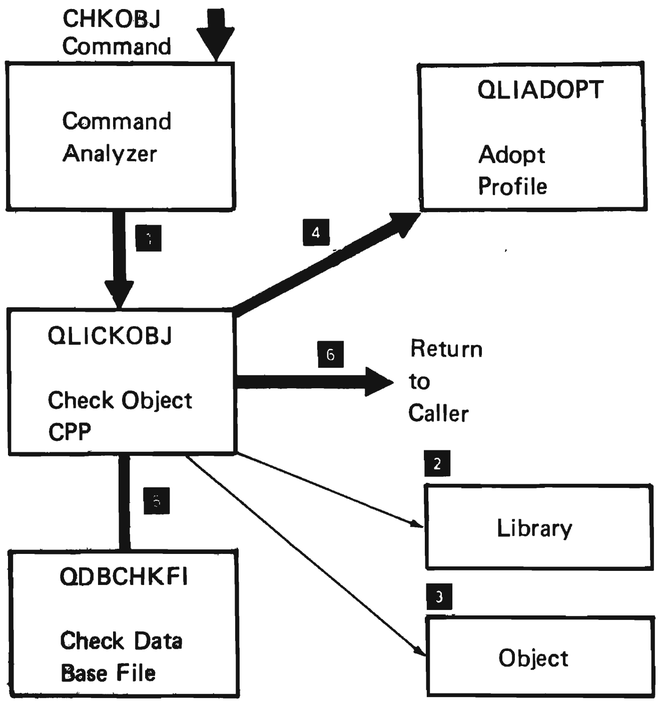


Figure LI-13. Check Object Command Overview

Add Library List Entry and Remove Library List Entry Commands

Figure LI-14 and the following text describe the operation of the Add Library List Entry (ADDLIBLE) and Remove Library List Entry (RMVLIBLE) commands.

1 The command analyzer decodes an Add Library List Entry or Remove Library List Entry command and passes control to QLICHLBL.

2 A library is added to or removed from the user portion of the process's library list.

3 Control is returned to the caller.

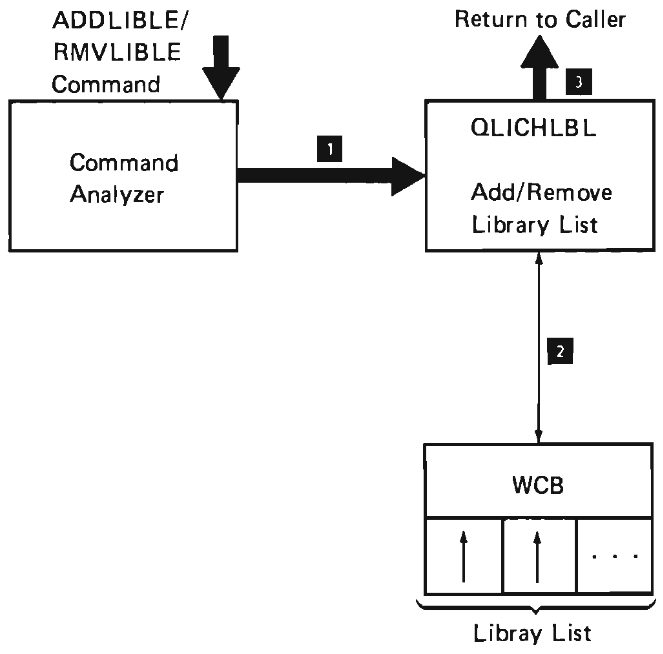


Figure LI-14. Add Library List/Remove Library List Command Overview

Create Duplicate Object Command

Figure LI-15 and the following text describe the operation of the Create Duplicate Object (CRTDUPOBJ) command.

1 The command analyzer decodes a Create Duplicate Object command and passes control to QLICRDUP.

2 QLICRDUP creates a duplicate object in a different library or with a different name in the same library as the original object. OIR records are duplicated as well as authorizations.

3 QLICRDUP calls QDBDUPF1 to duplicate data base files.

4 QLICRDUP calls QSPDUPQ to duplicate job queues and output queues.

5 QLICRDUP calls QMHDUPMQ to duplicate message queues.

6 QLICRDUP calls QLIDLOBJ to delete any duplicate objects it cannot complete.

7 Control is returned to the caller.

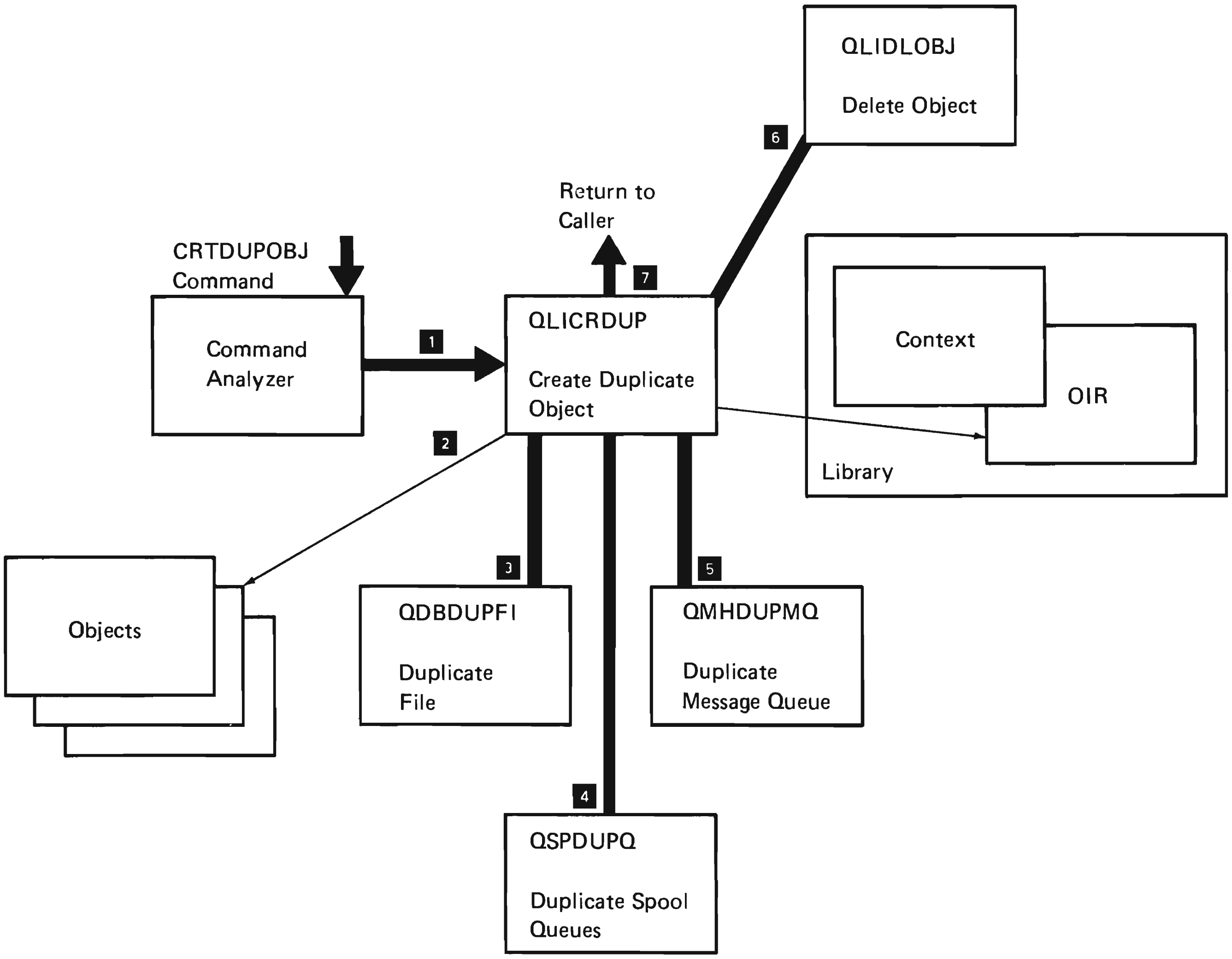


Figure LI-15. Create Duplicate Object Command Overview

Display Library List Command

Figure LI-16 and the following text describe the operation of the Display Library List (DSPLIBL) command.

1 The command analyzer decodes a Display Library List command and passes control to QLIDSPLL.

2 QLIDSPLL formats and prints the names, attributes, and text for the libraries on the process's library list, both system and user portion. The attribute field gives the type of the library and whether it is on the system or user portion of the list.

3 QLIDSPLL calls QLIDSLIB to format and display the names, attributes and text for the libraries on the process's library list, both system and user portions.

4 Control is returned to the caller.

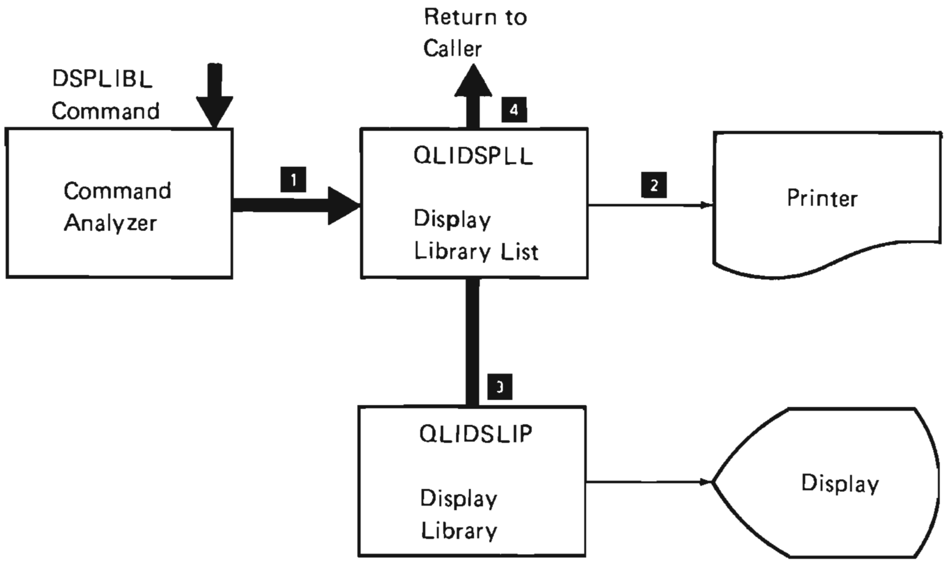


Figure LI-16. Display Library List Command Overview

Change System Library List Command

Figure LI-17 and the following text describe the operation of the Change System Library List command.

1. The command analyzer decodes the command, and control is transferred to QLICHLIB.
2. A library is added to or removed from the system portion of the process library list.
3. Control is returned to the caller.

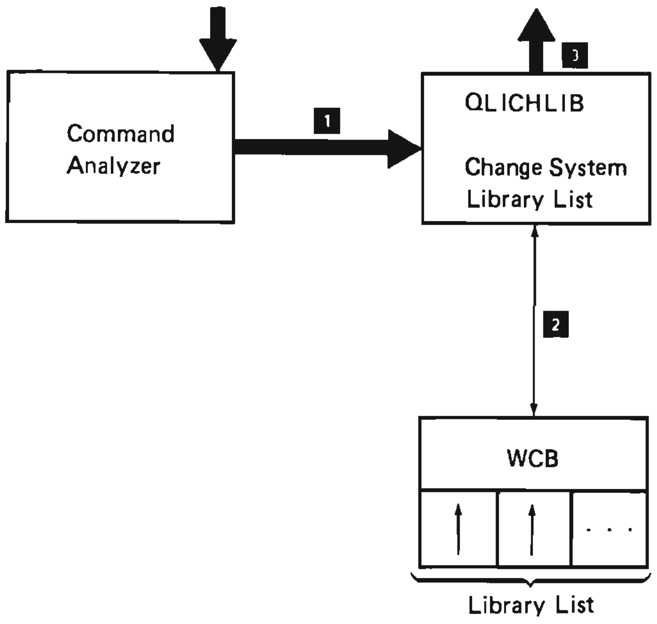


Figure LI-17. Change System Library List Overview

System Library Cleanup During Installation

Figure LI-18 and the following text describe the operation of the library cleanup function during installation.

1. Control is passed to WWLIVOIR from QINSTALL during installation.
2. The OIR of the system libraries QSYS, QSPL, QSRV, QGPL, and QRECOVERY are checked for marked damage. The damaged OIR is recovered if possible.
3. Control is returned to the caller.

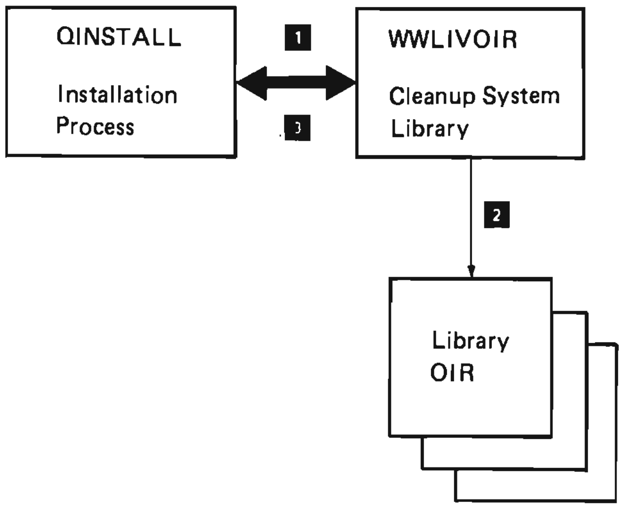


Figure LI-18. System Library Cleanup During Installation

Library Cleanup and Conversion During Reclaim Storage

Figure LI-19 and the following text describe the operation of the library cleanup and conversion during Reclaim Storage.

1 Control is passed to QLIRCLIB from QRCLAIM along with a list of all libraries on the system.

2 QRECOVERY is checked to see if any library recovery objects are in it. The presence of a library recovery object indicates that a library was not completely created.

3 QLIRCLIB deletes those libraries that are not completely created.

4 QLIRCLIB removes library recovery objects from QRECOVERY.

5 Each library on the list passed from QRCLAIM, plus its OIR and MCPI, is checked for marked damage. Damage recovery is performed, saving the information if possible.

6 Control is returned to QRCLAIM.

7 At some later time, determined by the libraries that have not been checked, control is again passed to QLIRCLIB, this time from QRCOMPST, from which a pointer to an individual library is passed. At this time the libraries passed to this module could include libraries which were previously not addressable through the machine context. These libraries have not yet been checked for damage. Only libraries not previously checked for damage are passed at this time.

8 QLICNVO is called to convert the library to the current release/modification level if needed.

9 If conversion is not required, the library, plus its OIR and MPCI, is checked for marked damage. Damage recovery is performed, saving the information if possible.

10 Control is returned to QRCOMPST.

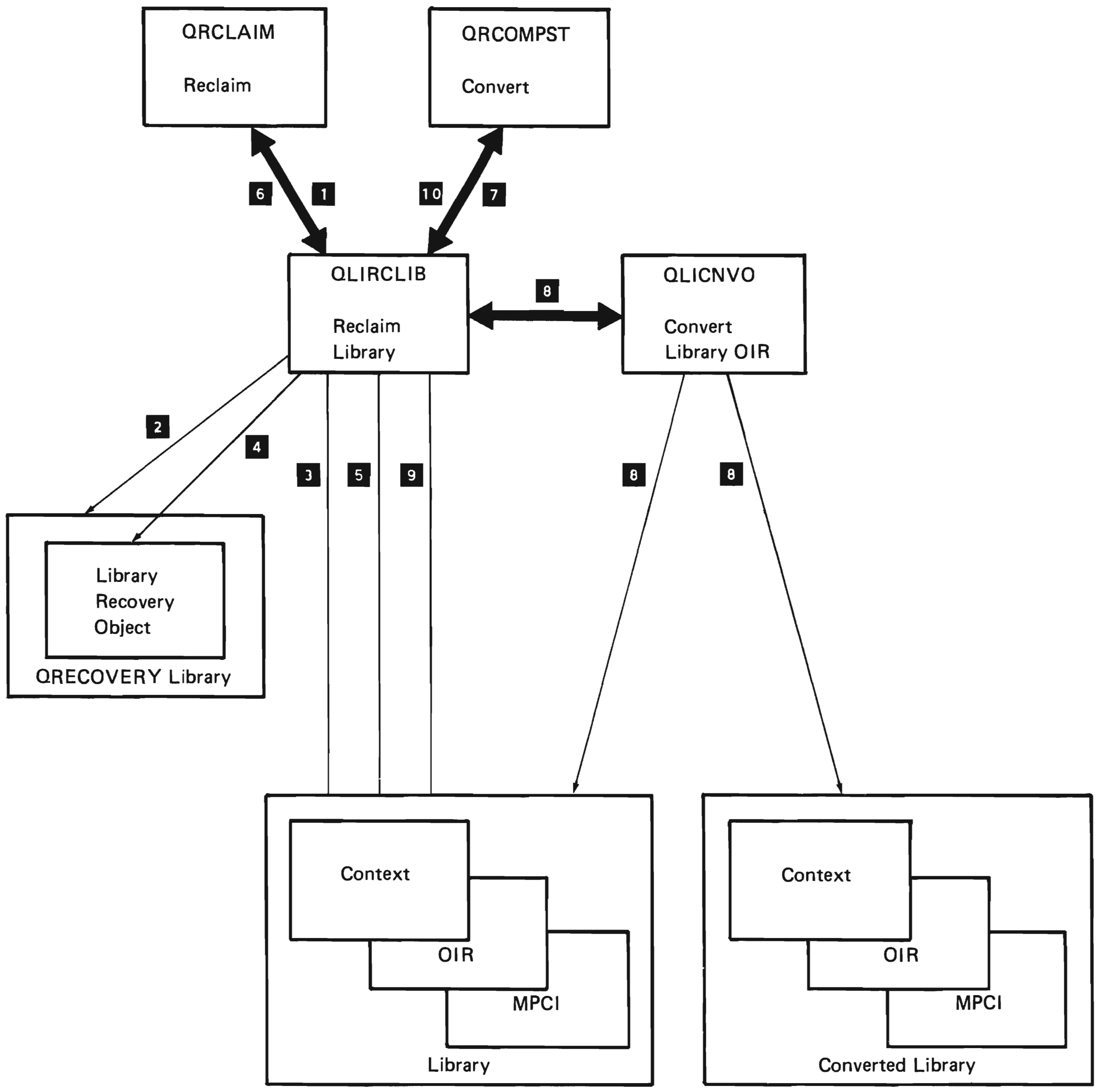


Figure LI-19. Library Cleanup and Conversion During Reclaim Storage

Library Conversion During Installation

Figure LI-20 and the following text describe the operation of the library conversion function during installation.

1. Control is passed to WWLICNVO from QINSTALL during installation.

2. Each library on the system is checked to ensure that it is at the current release/modification level. Any down level libraries are converted.

3. Control is returned to the caller.

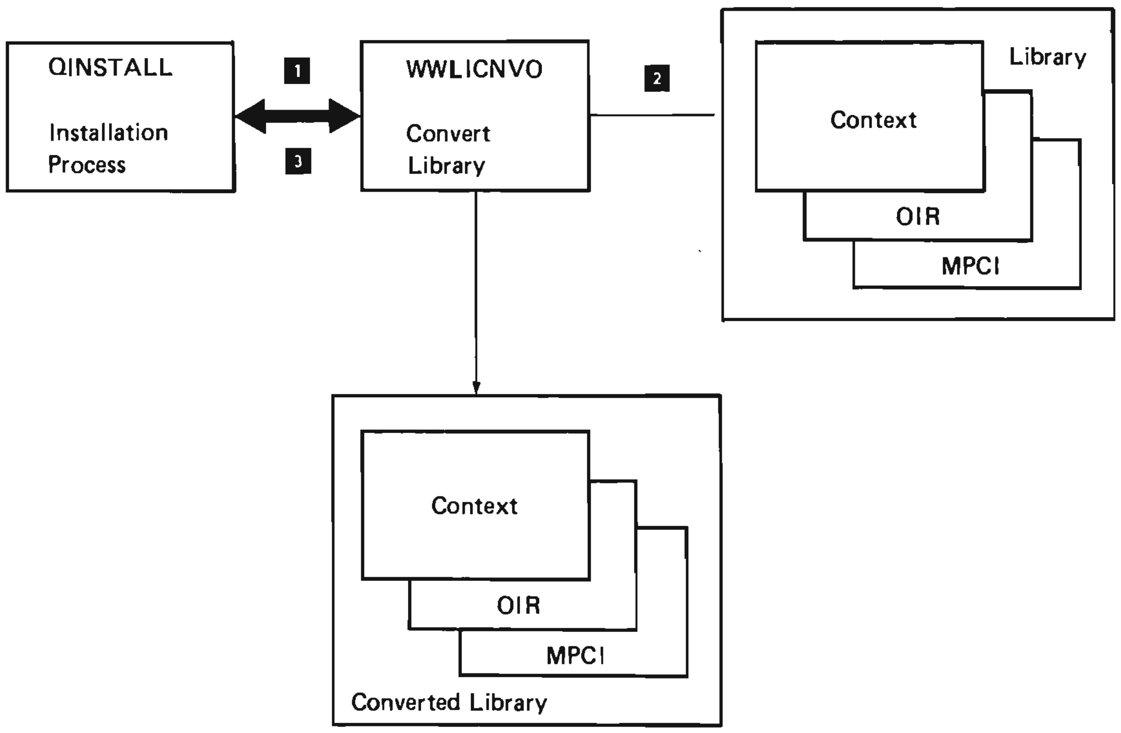


Figure LI-20. Library Conversion During Installation

INTRODUCTION

The message handler component of the CPF (control program facility) provides communications between the system operator and work stations, system users and programs, work stations and other work stations, and programs and the system. In order to supply those types of communications, the message handler provides the following functions:

- Message creation, storage, and retrieval
- Message routing and queuing
- Error detection and reporting
- Requester interface
- System log handling

Figure MH-1 shows an overview of the message handler functions.

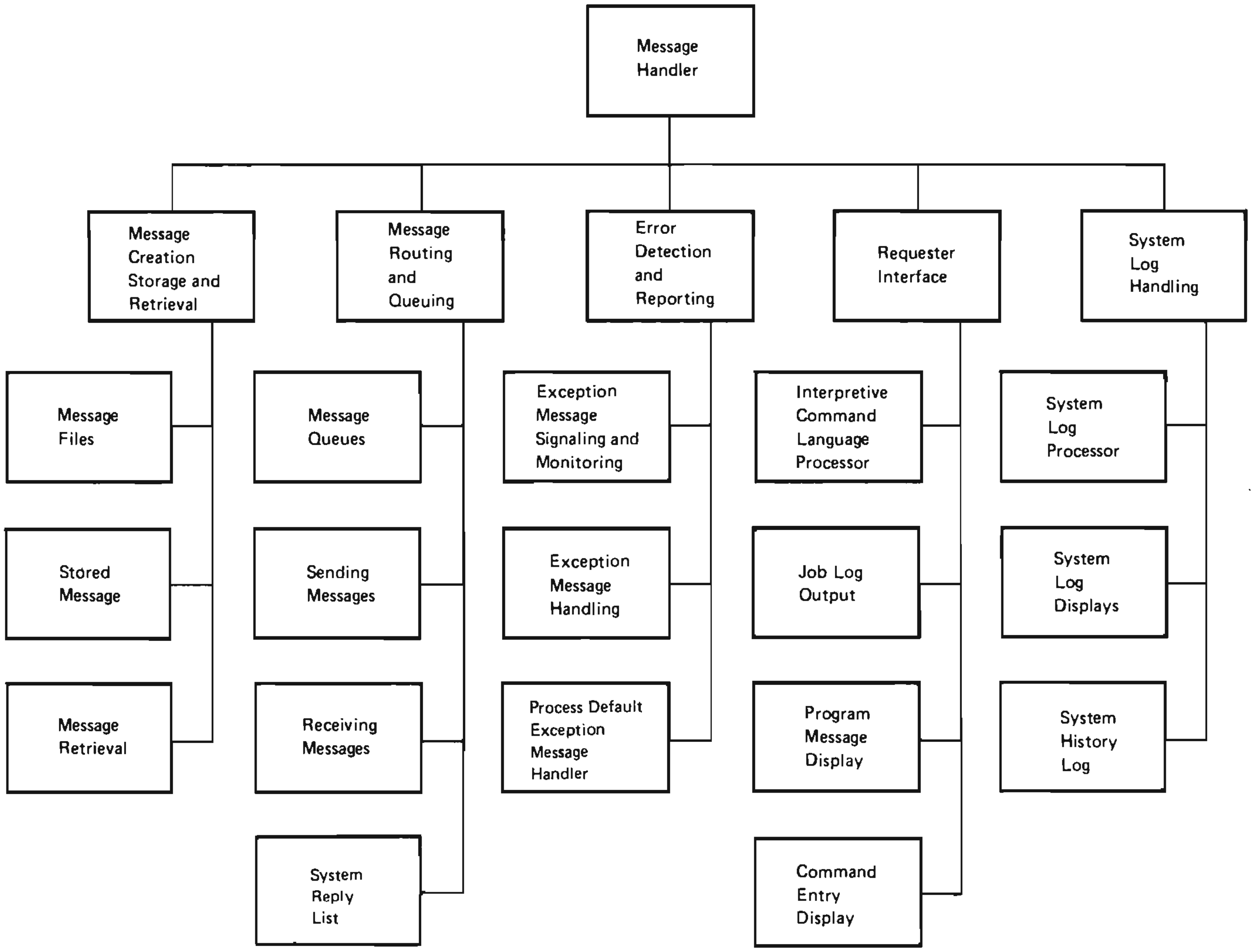


Figure MH-1. Message Handler Functional Overview

MESSAGE CREATION, STORAGE, AND RETRIEVAL

The two basic types of messages handled by the message handler components are predefined messages and impromptu messages. Predefined messages are stored in a message file and impromptu messages are messages whose text is generated at the time the message is sent.

Predefined messages are actually message descriptions that are stored in a message file under a message identifier. A message description can contain: first and second level message text, message severity, default reply, reply validity checking information, definition of variable data that can be sent with the message, and escape message handling and dump information.

When a stored message is sent, the message identifier and the message file name both must be specified. The message identifier is sent to the specified message queue or queues. When the message is received from the message queue, the receiver of the message can specify that he wants the text of the message returned. The message handler then retrieves the message text from the message file specified when the message was sent. The following modules support the interactive display of predefined stored messages:

QMHDSMSF–Display Message File: This module is the command processing program for the Display Message File (DSPMSGF) command, which supports displays of message ID, severity, and first level text, for a range of messages descriptions contained in a message file. If the output is to be printed, message descriptions in the specified range are located in the message file and are sent to print file QPMSGD. If the output is to be displayed on a screen, this module manages a subfile display of all message descriptions having IDs in the specified range.

QMHDSMSD–Display Message Descriptions: This module is the command processing program for the Display Message Description (DSPMSGD) command, which provides detailed displays of the message descriptions contained in a message file. Displays for both printer and interactive work station are supported.

QMHMSSFL–Message Subfile Manager: This module provides flexible support for the display of error messages to a message subfile record, for the message handler display management modules QMHDSMSD and QMHDSMSF.

Figure MH-2 and the following text describe message creation, storage, and retrieval functions.

1 QMHCRMSF–Create Message File: This module creates a stand-alone index with an associated space. The size of the space and extension attributes are determined by the size parameter of the Create Message File (CRTMSGF) command.

2 A message file is deleted without special processing. The librarian delete object function is called to delete message files.

3 QMHCRMSD–Create Message Description: This module adds messages to the message file. The message ID, severity, and offsets to text and control information are stored in the index portion. First level text, second level text, and message data formats, along with other data, are stored in the associated space.

4 QMHCHMSD–Change Message Description: This module changes a message description contained in a message file.

5 QMHDLMSD–Delete Message Description: This module removes all entries in the associated space for the message specified. When entries are removed, the space is made available for reuse and the index entry is deleted.

6 QMHMFSRH–Message File Search: This module builds a list of message files based on the message file overrides and it also searches the list of files for the requested message identifier. This module is invoked when a stored message is sent to obtain a system pointer to a message file based on the overrides at send time. The message file system pointer is stored as part of the message entry on the message queue. For messages on the message queues, the message file system pointer is used to retrieve the message. On other message queues the system pointer is used as a backup if the retrieve using the message file name fails.

7 QMHRTVMS–Retrieve Message CPP Interface: This module is the interface to the Retrieve Message (RTVMSG) command. After doing some preliminary command checking, QMHRTVMS calls QMHRTMSS to perform the retrieve message function.

8 QMHRTMSS–Retrieve Message Description: This module finds the specified message file and file override. The files are then searched in override order for the requested message identifier. Once found, the offsets contained in the index are used to find information about the message in the associated space. QMHRTMSS returns the requested message, substituting message data as replacement values in the message text.

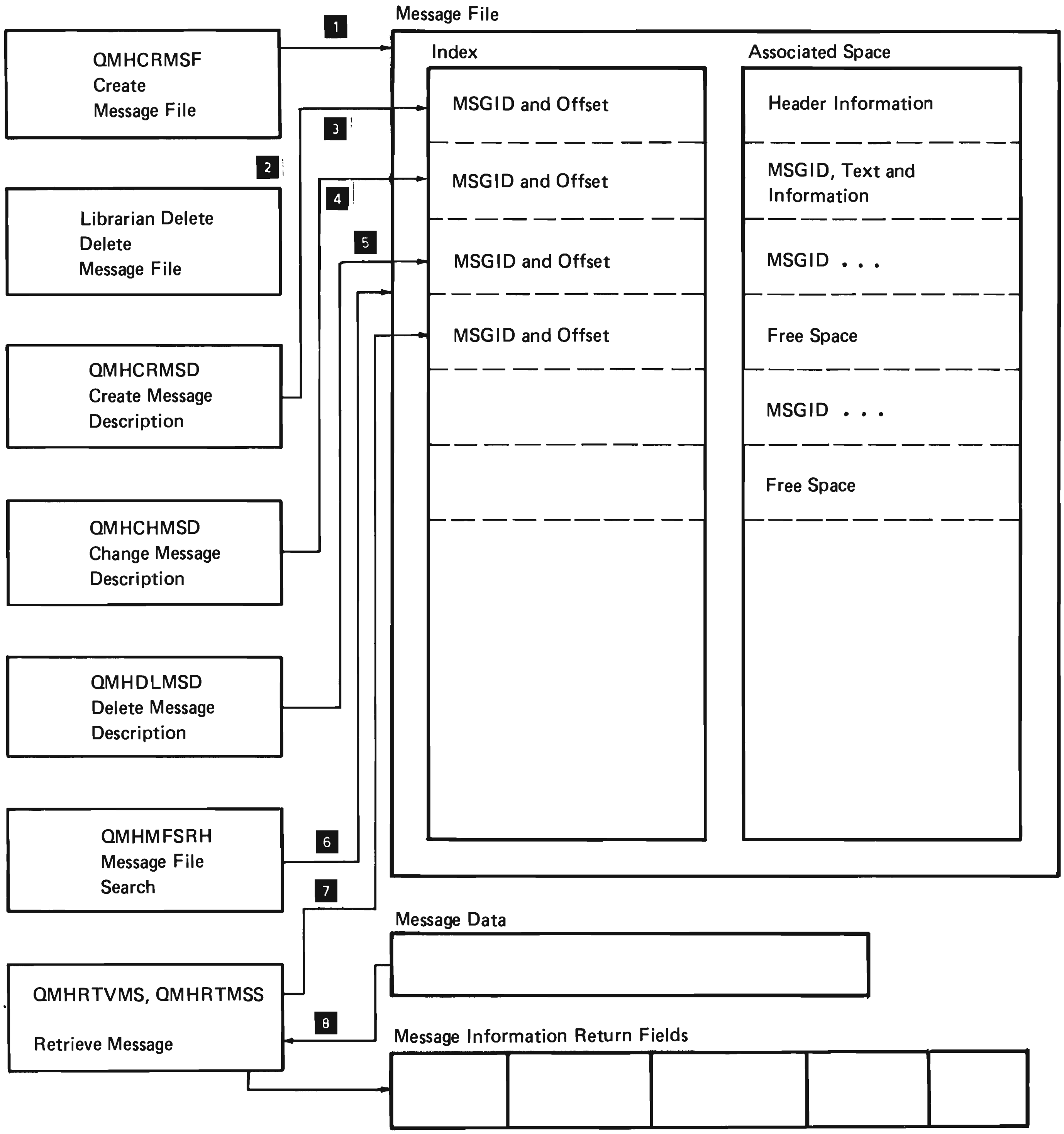


Figure MH-2. Message Creation, Storage, and Retrieval

MESSAGE ROUTING AND QUEUING

Message Routing

The valid message types are: request, scope, completion, diagnostic, exception, information, inquiry, reply, and sender's copy.

Request

This type of message requests a function to be performed. Generally request messages in CPF are commands.

Scope

This message specifies a program to be invoked when a program invocation of a job terminates processing. This type of message is used by CPF to back out certain environmental changes at the termination of the program invocation that requested the change.

Completion

The completion message provides information on the status of work performed.

Diagnostic and Exception

The diagnostic message provides information about input or processing errors. Generally, an exception message is also sent to indicate that diagnostic messages were sent.

There are three types of exception messages: escape, notify, and status.

- Escape: Notification of an error and unconditional surrender of control.
- Notify: Notification of an error or exceptional condition to which a reply can be sent. If unhandled, the default reply is sent and control is returned to the sender. If handled with an external handler, a reply can be sent; control is returned to the sender.
- Status: Notification of one program to a previous program of some condition but no message is placed on the message queue. If unhandled, control returns to the sender.

Information

This type of message provides general information.

Inquiry

This type of message requires a reply.

Reply

This type of message is an answer to inquiry or notify messages.

Sender Copy

This type of message is a copy of the inquiry or notify message. The copy is placed on the reply to message queue.

Message Queue Types

There are five types of message queues: user, work station, system operator, system log, and job. The message queues are space objects that contain a message queue header and a control entry, followed by variable length message entries or free space, or both.

User message queues are created, deleted, and accessed by using the command interface of the message handler component.

Each work station defined to the system has a message queue associated with it. This work station message queue is allocated to the same job or session as the work station. The work station message queue is created when its associated work station is defined to the system. The message queue is also deleted or renamed when its associated work station is deleted or renamed.

The system operator message queue is created by the system. It can be allocated by a user with system operator authority to a job, either at the console or a work station, by changing its delivery mode to break or notify.

The system creates a system log message queue associated with each system log (QHST, QCHG, QSVC). The Send Program Message (SNDPGMMSG) command, Send Message (SNDMSG) command, and ?SNDPMSG macro are used to send messages to the system log message queue. The Display Log (DSPLOG) command causes the messages in the system log message queue to move to the log file before they are displayed. The Delete Message Queue (DLTMSGQ) command deletes the system log message queues and automatically re-creates them.

A job message queue is created for each job. Each job message queue consists of a set of logical message queues (program message queues) which are created as needed. There is one program message queue for each active program invocation of a job to which a message has been sent, and one *EXT message queue which is used to communicate with the external requester of the job. The job message queue is the job log.

When the job ends, the messages in the job message queue are written to a spool output file for subsequent printing as the job log.

The following diagram shows the message type that can be issued for each message queue type.

	Message Queue Type				
Message Type	***EXT**	**Program**	**QSYSOPR**	**Work Station**	**User**
Information	X	X	X	X	X
Inquiry	X		X	X	X
Reply		X	X	X	X
Completion	X	X			
Diagnostic	X	X			
Escape/Exception		X			
Notify/Exception	X	X			
Request	X	X			
Scope	X	X			
Sender Copy		X	X	X	X
Status/Exception	X	X			

Message Queue Processing

Figure MH-3 and the following text describe message queuing operations.

1 QMHCRMSQ–Create Message Queue: This module processes the Create Message Queue (CRTMSGQ) command by creating the space for the message queue. QMHCRMSQ initializes the header with the queue name, type, size, and limits.

QMHDLMSQ–Delete Message Queue: This module processes the Delete Message Queue (DLTMSGQ) command. Before the message queue is deleted, an attempt is made to answer all outstanding inquiry messages. If errors occur, they are ignored and the queue is deleted.

QMHMODQS–Extend Message Queues: This module is called when a message queue or message file needs extending. The limits of extendability are user-defined by the size parameters of the create commands. This module adopts the QSYS user profile to assure authority to extend the space.

QMHALMSQ–Allocate Message Queue: This module is called by the Allocate Object (ALCOBJ) command to verify that message queues can be allocated. This is required because not all message queue types can be allocated. Work station message queues cannot be allocated by the allocate command; they are implicitly allocated wherever the work station is allocated. System log message queues can not be allocated.

2 QMHCHMSQ–Change Message Queue: This module processes the Change Message Queue (CHGMSGQ) command. Some of the attributes of a message queue that can be changed are the name of the break handling program, queue severity, and queue delivery mode. These attributes are all placed in the message queue header.

Send Message Processing

3 The following modules are used to place message entries on the message queue:

QMHSNMSG–Send Message (SNDMSG)[1]: This module does preliminary parameter checking and setup for the message queueing modules. If the message type is inquiry, this module calls QMHSNINQ to process the inquiry message. If the message type is informational, QMHSNSTQ is called to process the information message.

QMHSNPGM–Send Program Message (SNDPGMMSG)[1]: This module does preliminary parameter checking and setup for the message queueing modules. If the message type is inquiry or notify, QMHSNINQ is called; if the message type is status, QMHSNSTA is called. If the message type is not one of those types but is being sent to a program message queue, QMHSNJMQ is called. In any other case, QMHSNSTQ is called.

QMHSNBRK–Send Break Message (SNDBRKMSG)[1]: This module processes information messages to work station message queues. If the message type is inquiry, QMHSINQ is called to process the inquiry message.

QMHRSEXC–Send Resignal: This module is called via the ?RSGEXC macro. This module moves an escape message to the next program above the program issuing the resignal. If the next program is not monitoring for the message, QMHUNMSG is called to handle the unmonitored escape message condition.

QMHSNEVT–Send Event: This module sets up and signals an event for one of the following conditions:

- Break message sent via the Send Break Message command to an active work station
- Message sent to a queue in break or notify delivery mode

[1]This module is a CPP (command processing program).

- Message sent to a queue in wait for a message, because of a Receive Message command with wait
- Message sent to a system log, causing the log threshold to be exceeded

QMHSNSTQ–Send Message to Specified Queue: This module finds the specified message queue(s). The message is placed on the queue, and the control entry is updated. If an event needs to be signaled, QMHSNEVT is called to signal the event. An alert event is signaled if an alert message is sent to QHST while alert processing is active.

QMHSNINQ–Send Inquiry to Queue: This module finds the to message queue and the reply to message queue. A sender's copy message is placed on the reply to queue. The inquiry or notify message is placed on the to message queue. If an event should be signaled, QMHSNEVT is called to signal the event. QMHSNINQ also performs automatic reply processing with the INQMSGRPY job attribute and the system reply list.

QMHSNJMQ–Send Message: This module handles all message types, except for the inquiry to a program message queue. If the message is to be sent to a standard message queue, QMHSNSTQ is called. If the message type is escape, the receiving program exception monitors are examined for enqueuing and handling instructions. If an escape message is unmonitored, QMHUNMSG is called to perform problem analysis and send *FC. An alert event is signaled if an alert message is sent to QHST while alert processing is active.

QMHSNSTA–Send Status Message: This module handles the message if the message to be sent is a status message. If the message is being sent to a program that has a monitor to handle the message, an exception is signaled to that program. If not, control is returned to the sender and processing continues. If the message is sent to *EXT (directly or indirectly by way of PREV to top program) the status message is displayed on the error line and processing continues.

QMHSNRQ–Send Request: This module sends a request message to a job message queue. The request is always placed on the *EXT message queue. This is primarily used for spooled job input or by the submit job function.

QMHSNRPL–Send Reply: This module processes the Send Reply (SNDRPY) command. This function is used by programs to reply to inquiry and notify messages.

System Reply List

When an inquiry message is sent to a message queue and the inquiry message reply job attribute is set to system reply list INQMSGRPY (*SYSRPYL), the system reply list is searched for the same message ID. The system reply list entry specifies the message reply that is automatically sent to the message queue.

QMHCRTRL–Create Reply List: This module creates the system reply list space.

QMHADRLE–Add Reply List Entry: This module processes the Add Reply List Entry (ADDRPYLE) command. Reply list entries contain data for sequence numbers to be search ordered, a message identifier, reply text, dump indication, and compare data.

QMHCHRLE–Change Reply List Entry: This module processes the Change Reply List Entry (CHGRPYLE) command. All attributes of a reply list entry can be changed except the sequence number.

QMHRMRLE–Remove Reply List Entry: This module processes the Remove Reply List Entry (RMVRPYLE) command. Single entries or all entries can be specified for removal. If all entries are specified and the reply list is damaged, then the reply list is deleted and created again.

QMHDSPRL–Display Reply List: This module processes the Display Reply List (DSPRPYL) command. Entries are displayed or printed.

QMHRPYLO–Handle Display Reply List Options: This module is called to handle any reply list functions user-defined from the display.

Display Messages

4 QMHDSMSS–Display Messages: This module processes the Display Messages (DSPMSG) command, displays messages from the user, work station, and the system operator message queues. It will not display system log messages or messages on the job message queue. It is also used as the default program to handle break messages.

Receive Message Processing

5 QMHRCVMS–Receive Message Interface: This module is the interface to the Receive Message (RCVMSG) command. It does preliminary command checking, then calls QMHRCMSG to perform the receive function.

QMHRCMSS–Receive Message: This module receives the message from the message queue. Messages can be received by type or by key. Various information can be returned including resolved first and second-level text. There is also an option on the Receive Message command to remove the message after it has been received.

Move Message from One Program Queue to Another

6 QMHMVMS–Move Message: This module is invoked by the ?MOVPMSG macro. It moves diagnostic, escape, informational, and completion messages from the program invocation issuing the macro, to an invocation above it. Escape messages that are moved become diagnostic messages on the new program message queue.

Remove Message from Queue

7 QMHDLMS–Delete Message: This module processes the Remove Message (RMVMSG) command. It removes messages from a message queue. Messages can be removed individually by key or all old messages can be removed or all new messages can be removed or all messages can be removed.

Break/Notify Message Delivery

When a message is sent to a message queue in break or notify delivery mode, an event is signaled to the process that has the message queue allocated. The event is handled by the process control event handler which calls QMHDLVMS to handle the break or notify message delivery.

QMHDLVMS–Deliver Message: This module checks the message severity to determine if the severity satisfies a break or notify condition. This module also activates the work station and alarm if the message is to cause a notify condition. If the message is to cause a break condition, the program identified by the message queue is called to handle the break message. When the break or notify condition is complete, control is returned to the event handler and normal processing continues.

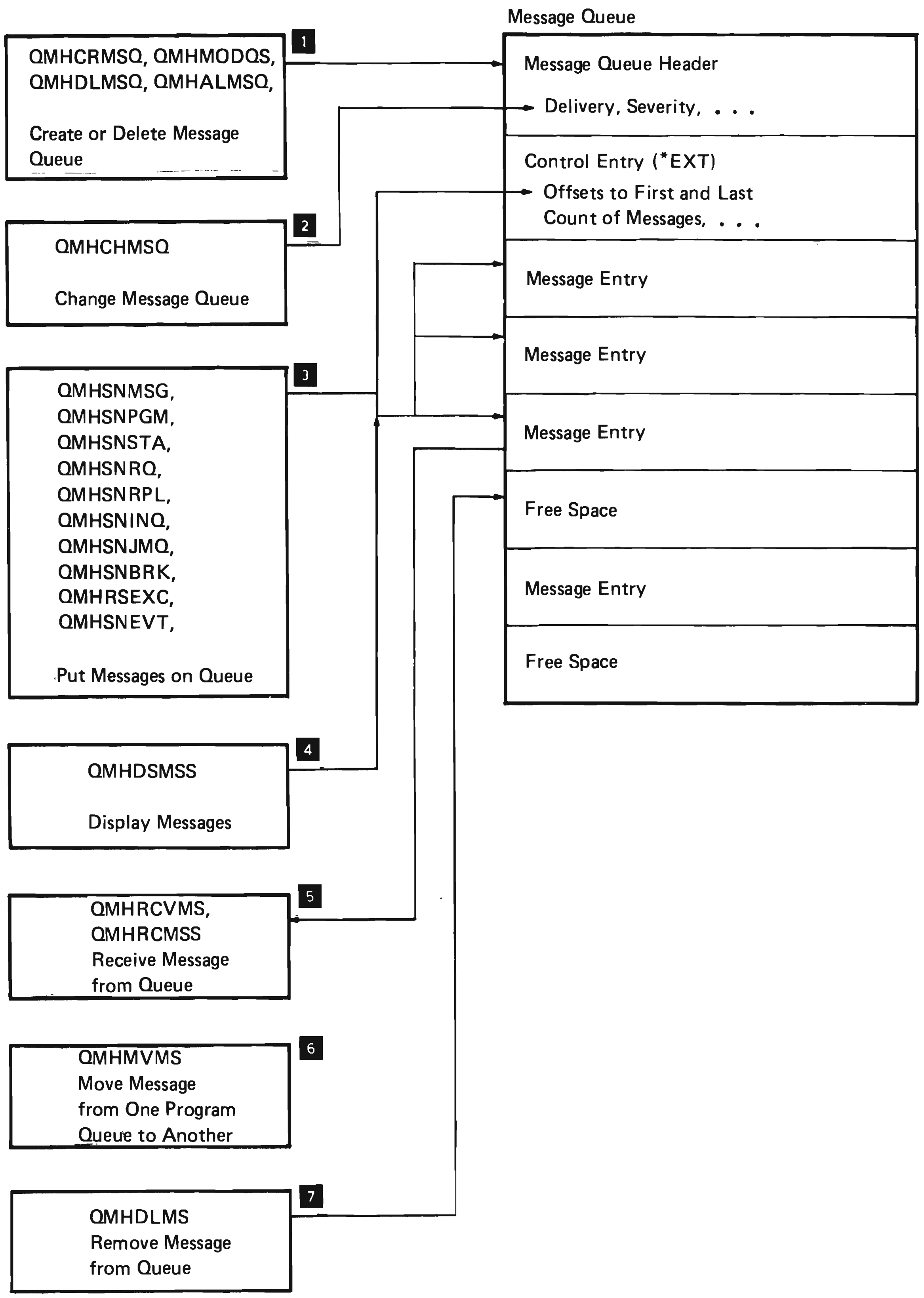

Message Queue
Message Queue Header
Delivery, Severity, . . .
Control Entry (*EXT)
Offsets to First and Last
Count of Messages, . . .
Message Entry
Message Entry
Message Entry
Free Space
Message Entry
Free Space
1
QMHCRMSQ, QMHMODQS, QMHDLMSQ, QMHALMSQ,
Create or Delete Message Queue
2
QMHCHMSQ
Change Message Queue
3
QMHSNMSG, QMHSNPGM, QMHSNSTA, QMHSNRQ, QMHSNRPL, QMHSNINQ, QMHSNJMQ, QMHSNBRK, QMHRSEXC, QMHSNEVT,
Put Messages on Queue
4
QMHDSMSS
Display Messages
5
QMHRCVMS, QMHRCMSS
Receive Message from Queue
6
QMHMVMS
Move Message from One Program Queue to Another
7
QMHDLMS
Remove Message from Queue

Figure MH-3. Message Queues

ERROR DETECTION AND REPORTING

Exception Messages

There are three exception message types: escape, notify, and status. The primary difference between these messages and others is that control can be transferred when an exception message is sent. In other words, the sender of the message may not get control back after sending the message.

Sending Exception Messages

Exception messages can be sent by the machine, by CPF, or by user programs. Messages sent by the machine are presented as MCHnnnn messages and indicate an exceptional condition was encountered by a System/38 instruction or function. Exception messages sent by CPF or user programs are sent by way of the send message interface.

Exception messages communicate status and conditions between programs within a job. As such, they are not normally sent to standard message queues (except as system information to the service or history log). Escape and notify messages are first placed on the job message queue; then an exception is signaled to the program invocation specified by the sender.

- Escape

  For escape messages, control will not return to the sender regardless of how the receiving program handles the escape message.

- Notify

  For notify messages, which are basically inquiry messages, control can return to the sender with a reply message available to the sender.

- Status

  For status messages, no messages are placed on the job message queue. If the receiving program does not monitor and handle a status message, control returns to the sender.

Monitoring Exception Messages

Exception messages can be *listened* for by declaring exception/message monitors. If an exception message is sent to a program with a corresponding monitor, control is directed as specified by the monitor. If there is no monitor, default system action is taken depending on the type of message sent.

Default System Error Handling

When exception messages are sent to programs that do not have a corresponding monitor, the default action is dependent on the type of message sent.

- Escape

  If an escape message is unmonitored, automatic problem determination is performed to take dumps, log the error condition to the service log, and call the default program, if appropriate.

- Notify

  If a notify message is unmonitored, ignored, or deferred, the default reply for the message is placed on the sending program's message queue and control is returned to the sender.

- Status

  If a status message is unmonitored, ignored, or deferred, control is returned to the sender. No message is placed on the job message queue, no dumps are taken, and no message is sent to the service log.

Exception Handling

The following modules handle exceptions:

QMHPDEH–Process Default Exception Handler: This module is invoked by the machine whenever there is an unmonitored machine interface exception. The primary functions of QMHPDEH are to convert raw machine interface exceptions to *MCHnnnn* messages. If the MCHnnnn message is unmonitored, QMHUNMSG is called.

QMHUNMSG–Unmonitored Message Handler: This module is invoked to handle unmonitored escape messages to programs. If problem determination is required, QMHAPD is called. The invocations stack is then searched until a program is found that is monitoring CPF9999. If no such program is found, the process is terminated.

QMHAPD–Automatic Problem Determination: This module is called by QMHPDEH when escape messages are unmonitored. It checks to see if logging and/or dumps are appropriate for the program receiving the exceptional condition. If dumps are appropriate, the priority of the job is lowered; and, if trace is active, trace is suspended while the dump is being taken. The default program, if specified on the message description, is called. When the dump has completed, trace is resumed. The priority of the job is reset by the scope handling program QMHRSTPR.

QMHRSTPR–Reset Priority: This command processing program is the scope handling program for message CPF2468, which is sent by QMHAPD. This module will reset the job priority that had previously been lowered by QMHAPD.

QMHRTNEX–Return From External Exception: This module is invoked only to return from an external exception handler. It verifies the type of handler and the type of exception. If the return is to the sender of a notify message, a reply is sent before control is returned to the sender.

Unmonitored Message Handling

Figures MH-4, MH-5, and MH-6 and the following text describe the handling of unmonitored messages.

1 Program B sends escape message CPF2450 to Program A. QMHSNJMQ is invoked to send the message.

2 Program A did not monitor for a CPF2450 message, so QMHUNMSG is invoked to handle the unmonitored escape message.

3 QMHAPD is invoked to send the error message to the service log and take dumps.

4 CPF9999 (the general function check message) is sent to Program A by QMHUNMSG.

5 Program A did not monitor for CPF9999 so QMHUNMSG sends the function check message to the invocation which called Program A, in this case QCL.

6 QCL has a monitor for CPF9999 and receives control to handle the exception.

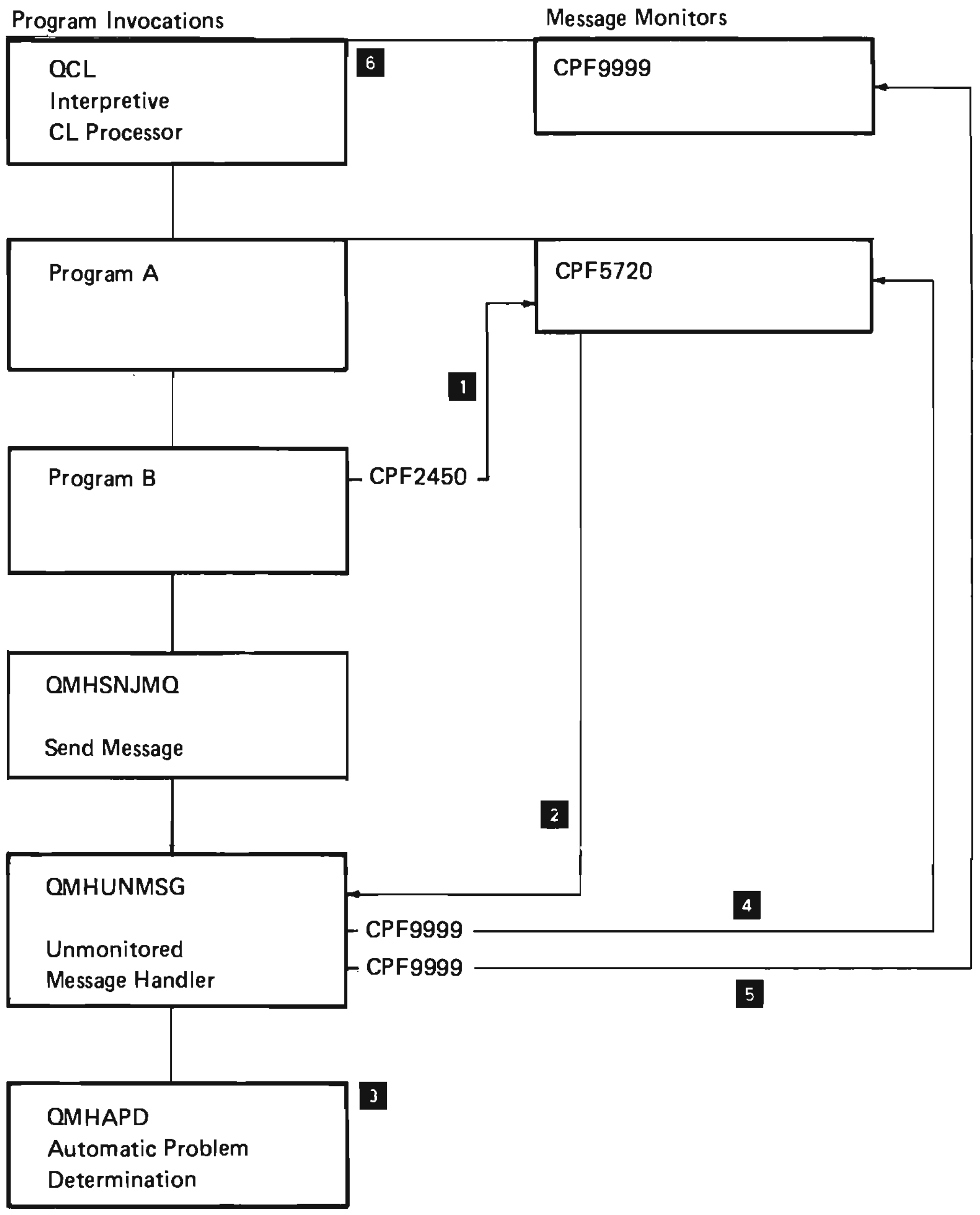


Figure MH-4. Unmonitored Escape Message

1 Program B sends notify message CPF2460 to Program A.

2 QMHSNINQ is invoked to send the notify message. Program A did not monitor for a CPF2460 so the default reply is sent.

3 Control is returned to Program B, the sender.

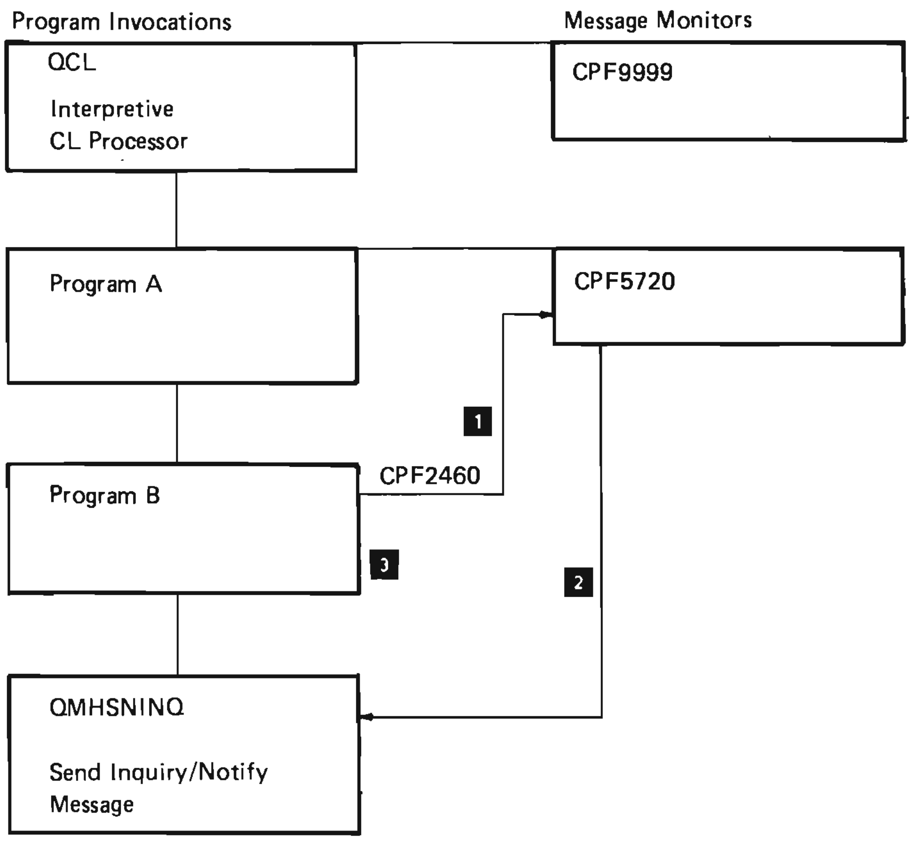


Figure MH-5. Unmonitored Notify Messages

1 Program B sends status message CPF2470 to Program A.

2 Program A did not monitor for a CPF2470 message so control is returned to Program B, the sender. No message is put on the message queue, no error is logged, and no dumps are taken.

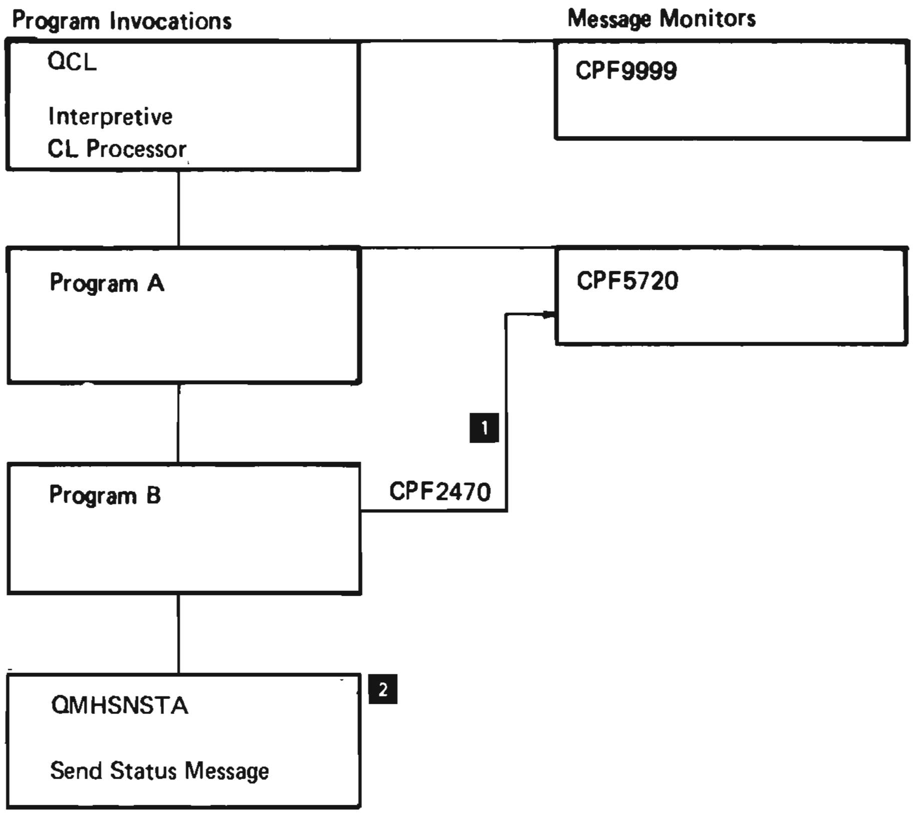


Figure MH-6. Unmonitored Status Message

This page is intentionally left blank.

REQUESTER INTERFACE

The requester interface consists of the command entry display, QCL (the interpretive command language processor), and the job message queue. QCL receives requests and commands from the job message queue and passes them to the command analyzer. In the batch environment, requests and commands are sent to the job message queue by the *spool* reader when the job is read in. In the interactive environment, commands and requests are entered by using the command entry display and are then sent to the job message queue. If specific criteria are met, then a message expansion display will be invoked.

The following are requestor interface modules:

QCL–Interpretive Command Language Processor: QCL is initially invoked when a process is initiated. If an initial program is specified in the user profile, QCL invokes it. QCL then receives a request message from *EXT and calls the command analyzer to process the request.

QMHGSD–General Session Display: This module presents the command entry display. Requests entered by way of the display are put on the job message queue. Requests already processed are displayed along with messages that may have been sent to the request processor. These requests and messages are the job log.

QMHJLOG–Write Job Log: This module is called to write the messages on the job message queue to a spool output file.

QMHFLTR–Job Log Filter: This module filters the previous request and its messages, when a new request is received, according to the logging and severity levels specified in the job description.

QMHCLOSE–Close QDGENDSP: This module is called to close the command entry display when a request receiving program terminates. It is invoked by QMHIREH (the invocation reference event handler).

QMHRPRQ–Replace Request: This module is called to replace the old request in the job message queue when prompting is requested and a new request is built by the prompter.

Initial Program Processing

When a job is initiated, QCL is invoked. QCL checks the user profile of the job; if there is an initial program specified, QCL gives it control. If there is no initial program or the initial program returns control to QCL, QCL goes on to receive a request message to process.

Program Message Display

Program message display allows a user to view the external message queue through the normal processing of a request. The processing of a request occurs in the interactive environment to allow the user access to messages.

The following module displays program messages:

QMHDSEXT–Display External Message Queue: This module displays the external message queue. Any informational or inquiry messages sent to the external program message queue from an interactive job can be displayed on the screen.

Interpretive Request Processing Overview

Figure MH-7 and the following text describe interactive request processing.

1 QCL–The interpretive CL processor issues a receive request message.

2 QMHRCMSS and QMHGSD–If no request message is in the job message queue (normal condition for interactive), the command entry display is presented for request entry. When a request is entered, it is put on the job message queue.

3 Receive message returns the request to QCL.

4 QCL calls the command analyzer to process the request.

5 The command analyzer syntax checks the CL and sends any diagnostic messages to QCL. If the command is executable, control is transferred to a command processing program.

6 Control is returned to QCL when the command processing program is complete. QCL then receives another request message which causes the command entry display to be presented again. This display will show the previous command with any messages sent to QCL and allow a new request to be entered.

7 QMHJLOG–When the work station operator signs off, the filtered job message queue messages are written to the job log.

8 QMHDSPJL–The user can display requests and related messages for a partially completed job by using the DSPJOBLOG command or selecting the Display Job Log option from the Display Job command.

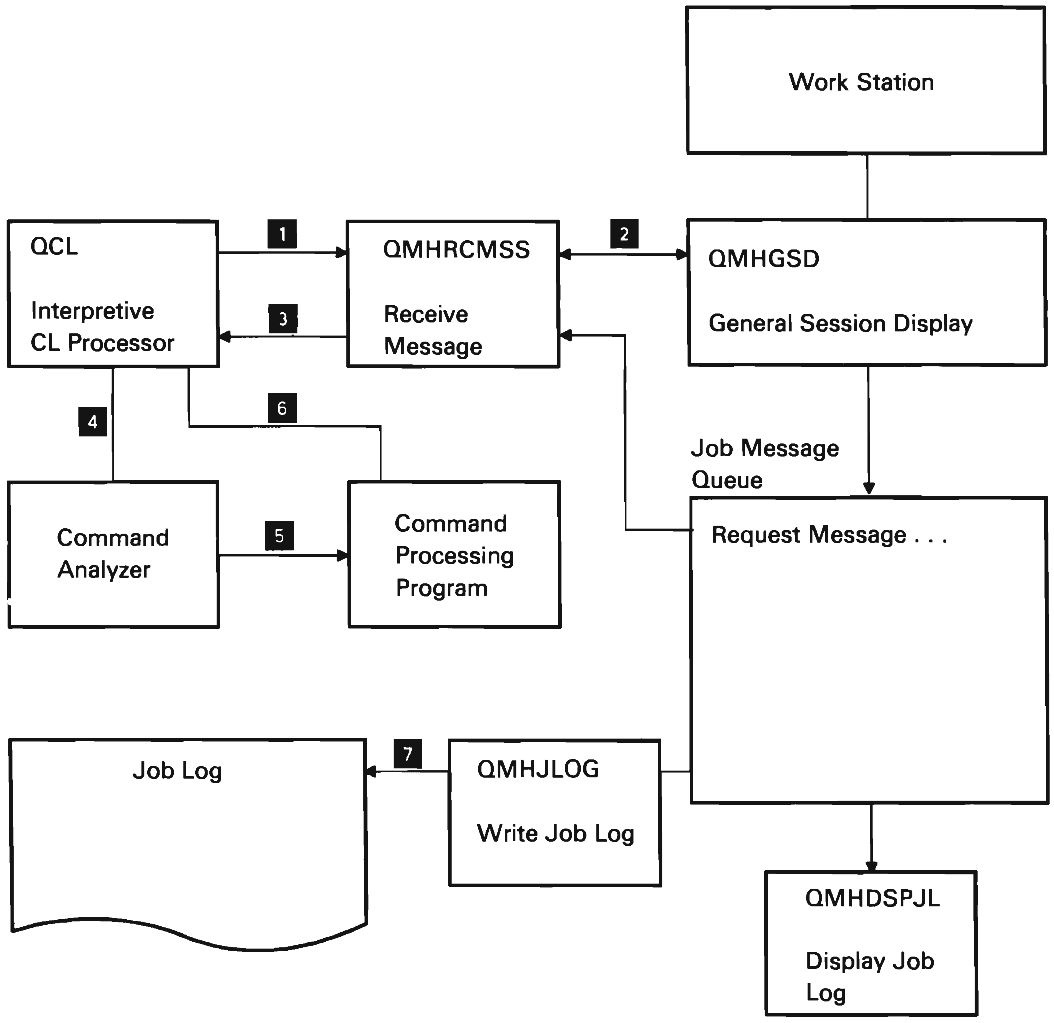


Figure MH-7. Request Processing Interactive

Batch Request Processing Overview

Figure MH-8 and the following text describe batch request processing.

1 The spooling reader process reads the job input stream, syntax checks it, and then sends a message to the job message queue.

2 The batch monitor starts QCL. It receives a request message from the job message queue and calls the command analyzer.

3 The command analyzer and the command processing programs can send or receive messages from the program queues.

4 If there are no more request messages on the job message queue, QCL terminates the process.

5 Filtered messages are sent to the job log when the process is terminated.

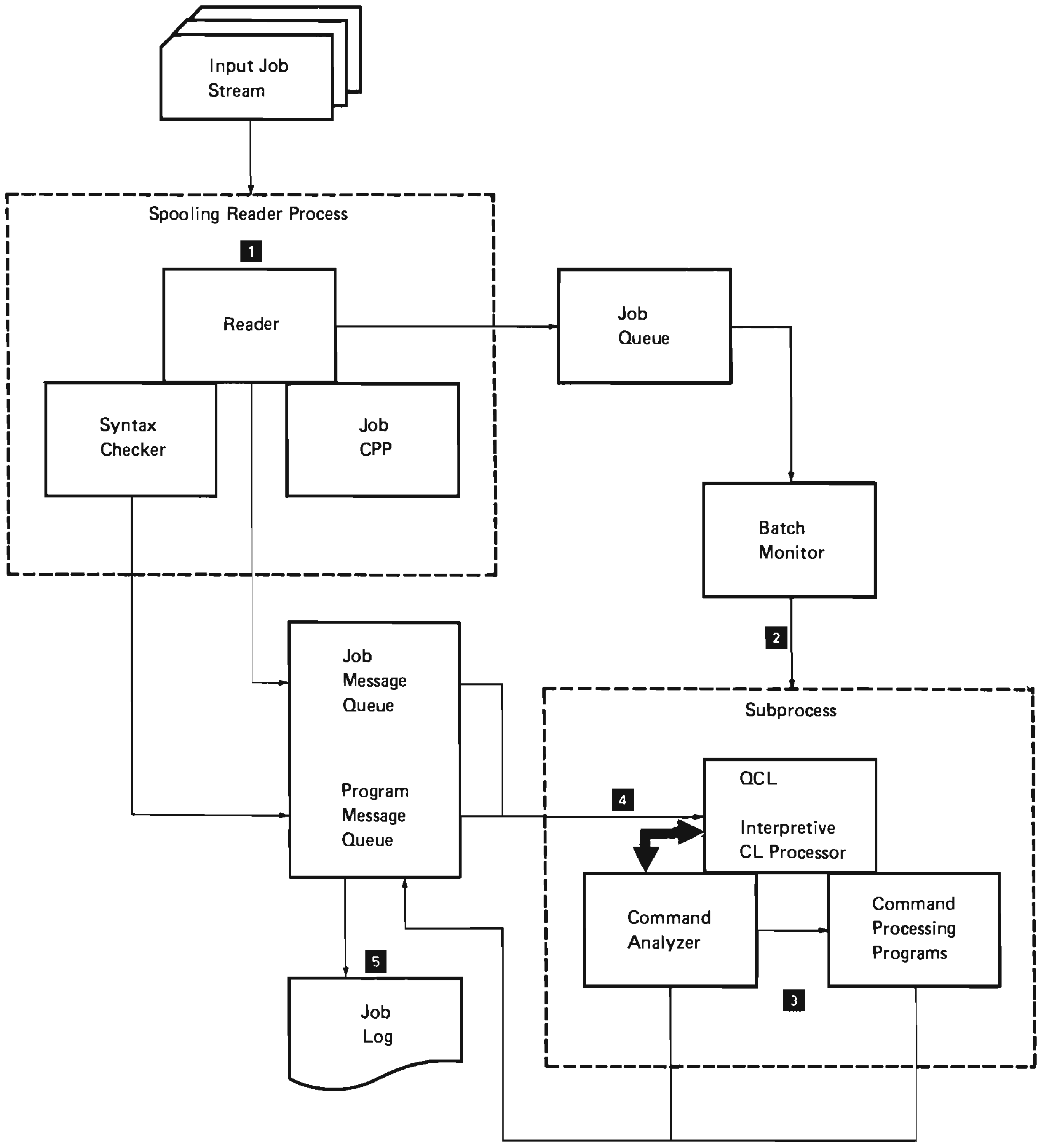


Figure MH-8. Batch Request Processing Overview

SCOPE MESSAGE PROCESSING

When a scope message is sent to an invocation, the invocation reference event is set to be signaled when the invocation terminates. A scope message is placed on the program message queue of the invocation, and may be associated with either the program invocation or with the invocation level. When the invocation terminates, the event is signaled and QMHIREH is invoked.

QMHIREH–Invocation Reference Event Handler: This module gets control when the invocation reference event is signaled. If an invocation was terminated by a transfer of control, any scope messages associated with the invocation level are moved to the program message queue of the invocation to which control was transferred. For all remaining scope messages, the program specified in the scope message is called. As each scope message is handled, it is removed. When all scope messages have been processed, control exits the event handler and processing continues.

SYSTEM LOGS

System Log Structure And Processing

System logs consist of system log message queues and data base files called log versions. The message queue is used as a temporary repository for messages until there is a sufficient number to justify writing them to a log version. During start CPF a check is made to ensure all the system log message queues are available and undamaged. An event monitor is created under the start CPF process to listen for the write system logs to data base event. The modules to handle system logging are:

QMHLINIT–System Logging Initializer: This module is invoked during IPL processing. It assures the system log message queues are available and undamaged. When the number of messages sent to a system log message queue reaches a predefined threshold or a system log is displayed, an event is signaled to indicate the messages in the system log are to be written to a log version. An event description is set up by QMHLINIT to monitor for this event.

QMHLOGER–Logger: This module is an event handler and is invoked when the write system logs event is signaled. All entries in the system log message queue are written to a log version and removed from the message queue. When the queue is empty, the module ends.

QMHCLVER–Create Log Version: This module sends a message to the system operator to inform him of the full log version and creates a new log version.

System Log Display

The following module handles system log displays:

QMHLDISP–Display Log: This module processes the Display Log (DSPLOG) command. Before a system log is displayed, QMHLDISP signals an event to have all current messages in the system log message queue sent to the log version. When logging is complete, the log versions are searched to find the log entries requested.

Menu

INTRODUCTION

The menu component of the CPF (control program facility) provides six menu displays that interface with the CPF. The menus are as follows:

- Program call menu
- Command selection menus
- Configuration menu
- System operator menu
- Programmer menu
- System request menu

Program Call Menu

The program call menu supports the execution of four commands:

- Call Program (CALL)
- Display Messages (DSPMSG)
- Send Message (SNDMSG)
- Sign-Off (SIGNOFF)

A call to QCALLMENU will cause the menu to be displayed. The QCALLMENU program is the initial program for the QUSER profile provided with the system.

Command Selection Menus

The command selection menus provide access to lists of command names and command selection menu names grouped by verb or subject functions.

Command Grouping Menu

The default menu, the command grouping menu, displays a list of command selection menus that identify major functions available on the System/38. The default menu is displayed when prompting is requested without entering a command name, when a question mark is entered, or when the Display Menu (DSPMNU) command is entered or selected off the programmer or system operator menu without specifying a menu name. An option may be selected, a command name or a menu name may be entered, or up to 10 characters of a command name may be specified to obtain a list of partial command names. Succeeding displays depend on the option selected from the command grouping menu.

Configuration Menu

The configuration menu supports the execution of a group of commands during the start CPF process or during the execution of the concurrent service monitor component. If the configuration option is specified on the start CPF prompt or on the service function menu, the configuration menu is displayed. Command entry is limited to the commands shown on the display.

System Operator Menu

The system operator menu supports the execution of three commands through function keys, 13 other commands via option number, and general command entry through option 5. The menu is the initial program for the QSYSOPR user profile. A call to QOPRMENU program will cause the menu to be displayed.

Programmer Menu

The programmer menu supports the execution of the display message command through CF6, ten options corresponding to several commands commonly used by programmers, and general command entry through option 5. Several of the options result in an asynchronous batch job submission. Option 7 supports a display of the jobs submitted from the menu. The menu is supported on the 24x80 display only.

The menu is the initial program for the QPGMR user profile. A call to QPGMMENU program will cause the menu to be displayed.

System Request Menu

The system request menu supports the execution of five commands and through option 1 provides the function of transferring to a secondary interactive job. The menu is displayed with the system request key.

GENERAL OVERVIEW

Menu Modules

The menu component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QCALLMENU–Program Call Menu: This module displays the program call menu and processes the function selected.

-->QMNCMDPM–Command Selection Menu Facility: This module displays the command selection menus and processes commands through the command analyzer. QMNCMDPM is also the command processing program for the Display Menu (DSPMNU) command.

-->QMNCONFG–Configuration Controller and Menu: This module displays the configuration menu and processes any of the commands selected.

-->QPGMMENU–Programmer Menu: This module displays the programmer menu and processing of the commands selected.

-->QMNSYSRQ–System Request Menu: This module displays the system request menu and processes any of the commands selected.

-->QOPRMENU–System Operator Menu: This module displays the system operator menu and processes any of the commands selected.

The following module is only used within the menu component:

QMNTXTBL–Build Menu Text Space Objects: This module is used to build text for the command selection menus and configuration menu displays.

Program Call Menu Overview

Figure MN-1 describes the function of the program call menu.

1 QCALLMENU is a program without any parameters and may be invoked as the initial program in a user profile or by the Call (CALL) command. QCALLMENU is coded in CL to serve as a sample menu program.

2 The program call menu is shown on the user display by using the QMNCALLM display device file and the Send/Receive File (SNDRCVF) command. The user selects the desired option and enters any necessary parameters.

3 Depending on the option selected, one of the following is executed:

A Option 1: Call QCAEXEC, the high-level language interface module of command analyzer, to execute a call program. If the program name is not qualified and no parameters are used, the program is called directly without the use of QCAEXEC.

B Option 2: Execute the Display Messages (DSPMSG) command.

C Option 3: Execute the Send Message to System Operator (SNDMSG) command.

D Option 90: Execute the Sign-Off (SIGNOFF) command.

4 If an exception message is sent as a result of executing the user-defined option, the exception message is displayed in the error message line at the bottom of the program call menu.

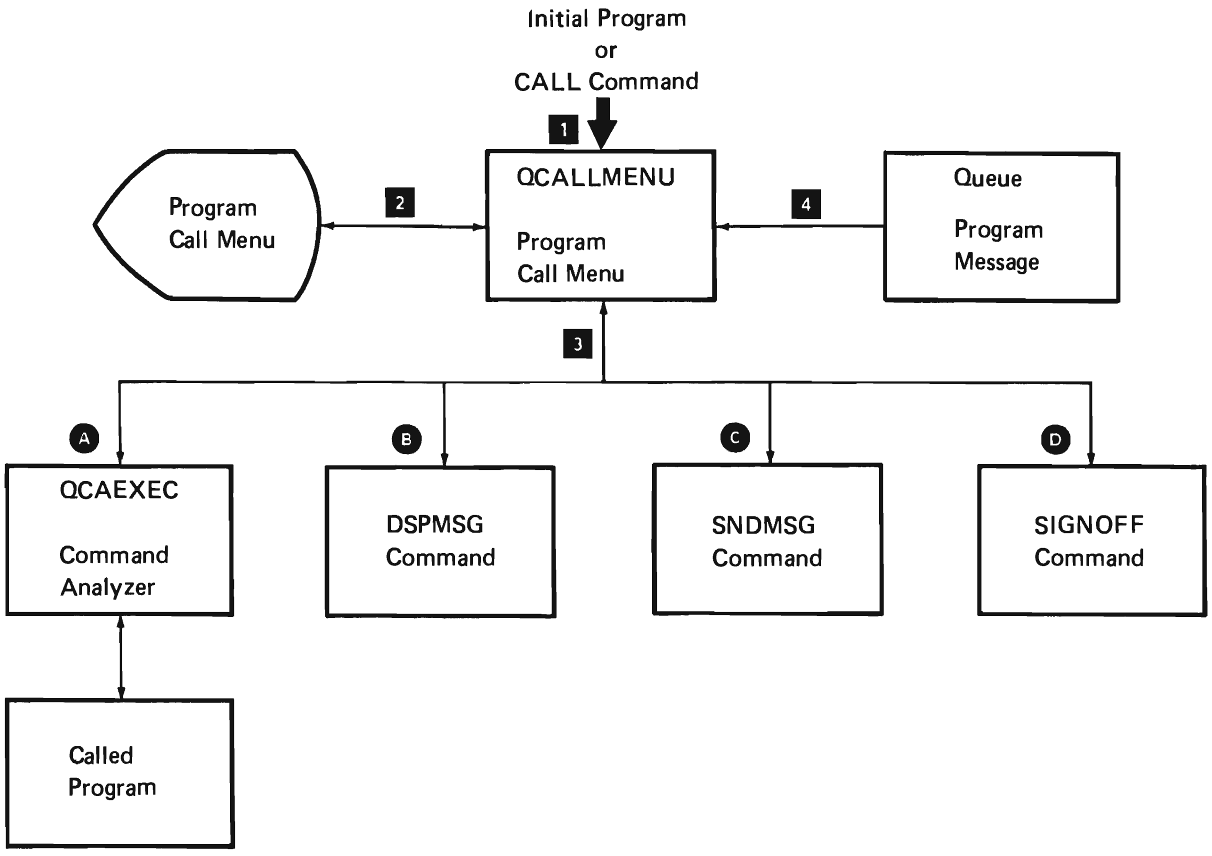


Figure MN-1. Program Call Menu Overview

Command Selection Menu Overview

Figure MN-2 and the following text describe the function of the command selection menus.

1 QMNCMDPM is called when prompting is requested without entering a command name, when a question mark is entered, or when the Display Menu (DSPMNU) command is entered or selected off the programmer or system operator menu. The command analyzer may have been called for interactive command execution or for CL source entry.

2 The command selection menu is built using an index object, QMNCMDPM, that contains the text for all command selection menus.

3 Each line of the command selection menu is sent to a subfile in the QMNCMDPM display device file. After all lines have been sent, they are displayed. The user may select an option, enter a command name or a command selection menu name, or specify up to 10 characters of a command name to obtain a list of partial command names. The command name or command selection menu name does not have to be one of those on the command selection menu.

4 The command selection menu is built according to the user-defined inputs and options.

5 If a command is selected, the command name is passed to the command analyzer for further processing. If a command selection menu is selected, the command selection menu is built using QMNCMDPM.

6 Control is returned to QMNCMDPM if a command selection menu was selected or QMNCMDPM was called via the Display Menu command. Otherwise, control is returned to the caller.

7 The command analyzer then calls the prompter to prompt for any required parameters that are missing (unless the CF16 key was pressed at the command selection menu). Depending on the environment, the command analyzer may call the command processing program to execute the command.

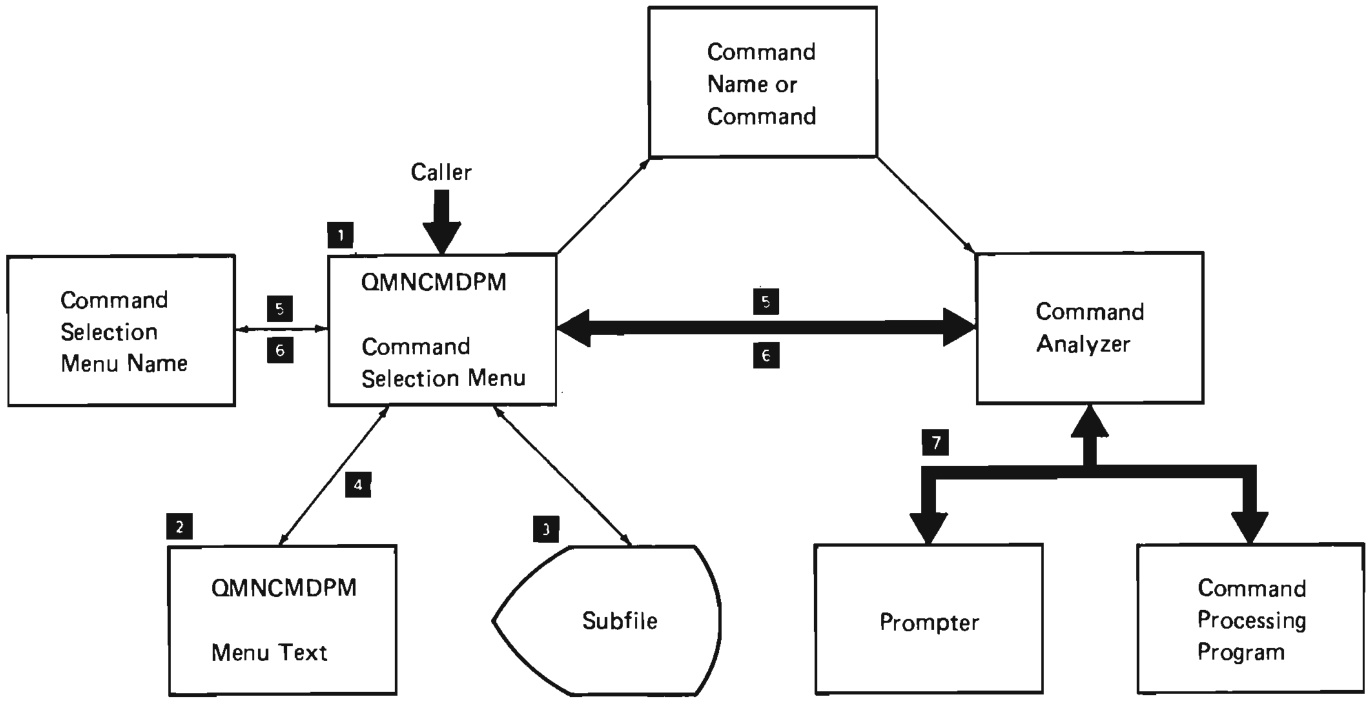


Figure MN-2. Command Selection Menu Overview

Configuration Menu Overview

Figure MN-3 and the following text describe the functions of the configuration menu.

1 QMNCONFG is called by the start CPF program when configuration is requested on the start CPF prompt display or by the concurrent service monitor when the configuration option is selected.

2 The configuration menu is built using a space object, QMNCONFG, that contains the menu text.

3 Each line of the configuration menu is sent to a subfile in the QMNCONFG display device file. After all lines have been sent, they are displayed. The user then selects the command name of the command to be executed. The command processing program for the command selected must be one of those on the configuration menu. A command other than one on the menu cannot be entered.

4 QMNCONFG calls the command analyzer and passes to it the command name or command along with the prompt/no prompt and execute options. The command analyzer validity checks the command, optionally calls the prompter to prompt for any required parameters that are missing, and then calls the appropriate CPP to execute the command.

5 If an exception message is sent as the result of executing the user-defined command, the exception message is displayed in the error message line at the bottom of the configuration menu.

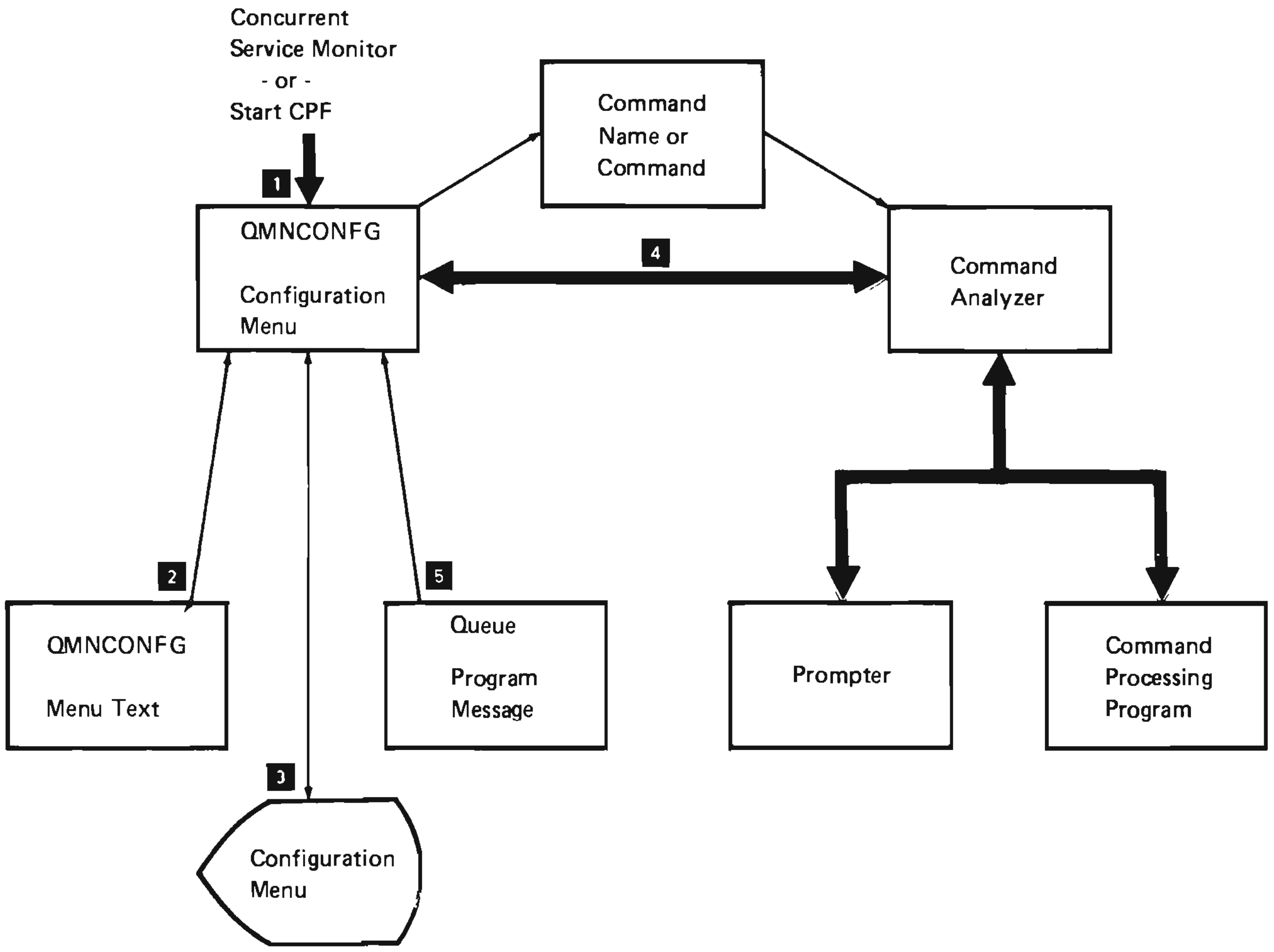


Figure MN-3. Configuration Menu Overview

System Operator Menu Overview

Figure MN-4 and the following text describe the functions of the operator menu.

1 QOPRMENU is called by QCL when it is an initial program, or it is invoked as a result of a call command.

2 The system operator menu is displayed, and the user selects an option.

3 When option 5, execute a command, is selected QOPRMENU calls the command analyzer and passes the command to it, along with the prompt/no prompt and execute options.

The command analyzer validity checks the command, optionally calls the prompter, and then calls the appropriate command processing program to execute the command. When finished, the prompted string is returned to the command line of the menu, if prompting was requested.

When option 6, submit job, is selected and the CF4 key is pressed, QOPRMENU calls the command analyzer to prompt for the command, and the request data. The command is not executed; instead, the request data is submitted as a batch job. If the CF4 key is not pressed, and the enter key is pressed, the request data is syntax checked by the command analyzer before it is submitted as a batch job. The CF15 key allows submitting jobs with syntax errors.

4 For all other options, QOPRMENU calls the command processing program directly.

5 If executing the command causes an exception message to be sent, the exception message is displayed in the error message line at the bottom of the system operator menu.

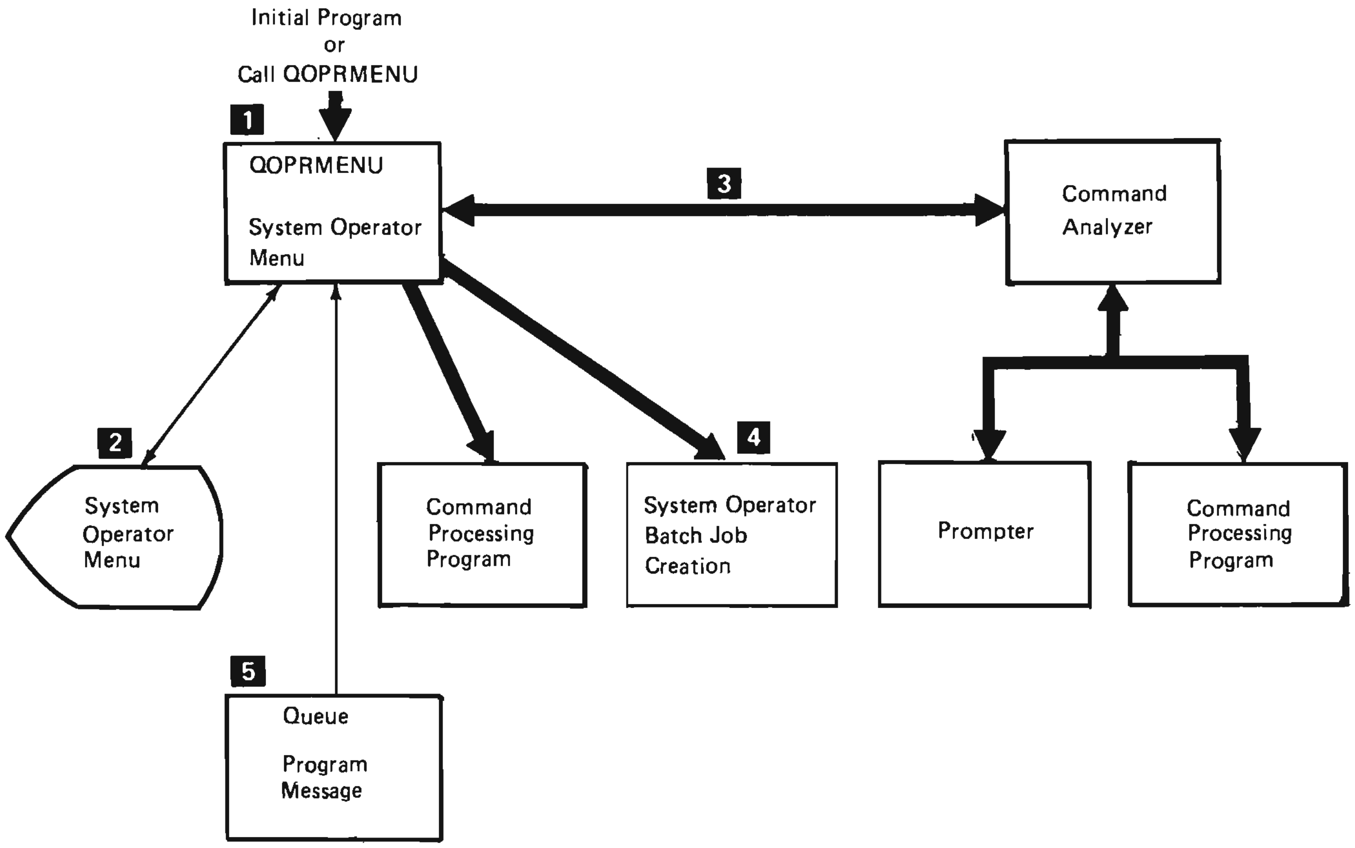

Initial Program or Call QOPRMENU
1
QOPRMENU
System Operator Menu
3
Command Analyzer
2
System Operator Menu
Command Processing Program
4
System Operator Batch Job Creation
Prompter
Command Processing Program
5
Queue
Program Message

Figure MN-4. System Operator Menu Overview

Programmer Menu Overview

Figure MN-5 and the following text describe the function of the programmer menu.

1 QPGMMENU is called when it is an initial program, it is invoked as a result of a call command or it is invoked by the command display program menu (DSPPGMNNU).

2 The programmer menu is displayed, and the user selects an option.

3 When option 5, execute a command, is selected QPGMMENU calls the command analyzer and passes the command to it, along with the prompt/no prompt and execute options. The command analyzer validity checks the command, optionally calls the prompter, and then calls the appropriate command processing program to execute the command. When finished, the prompted string is returned to the command line of the menu, if prompting was requested.

When option 6, submit job, is selected and the CF4 key is pressed, QOPRMENU calls the command analyzer to prompt for the command, and the request data. The command is not executed; instead, the request data is submitted as a batch job. If the CF4 key is not pressed, and the enter key is pressed, the request data is syntax checked by the command analyzer before it is submitted as a batch job. The CF15 key allows submitting jobs with syntax errors.

4 For the create command, QPGMMENU submits a batch job to perform the create function requested.

5 For all other options QPGMMENU calls the command processing program directly.

6 If executing the command causes an exception message to be sent, the exception message is displayed in the error message line at the bottom of the programmer menu.

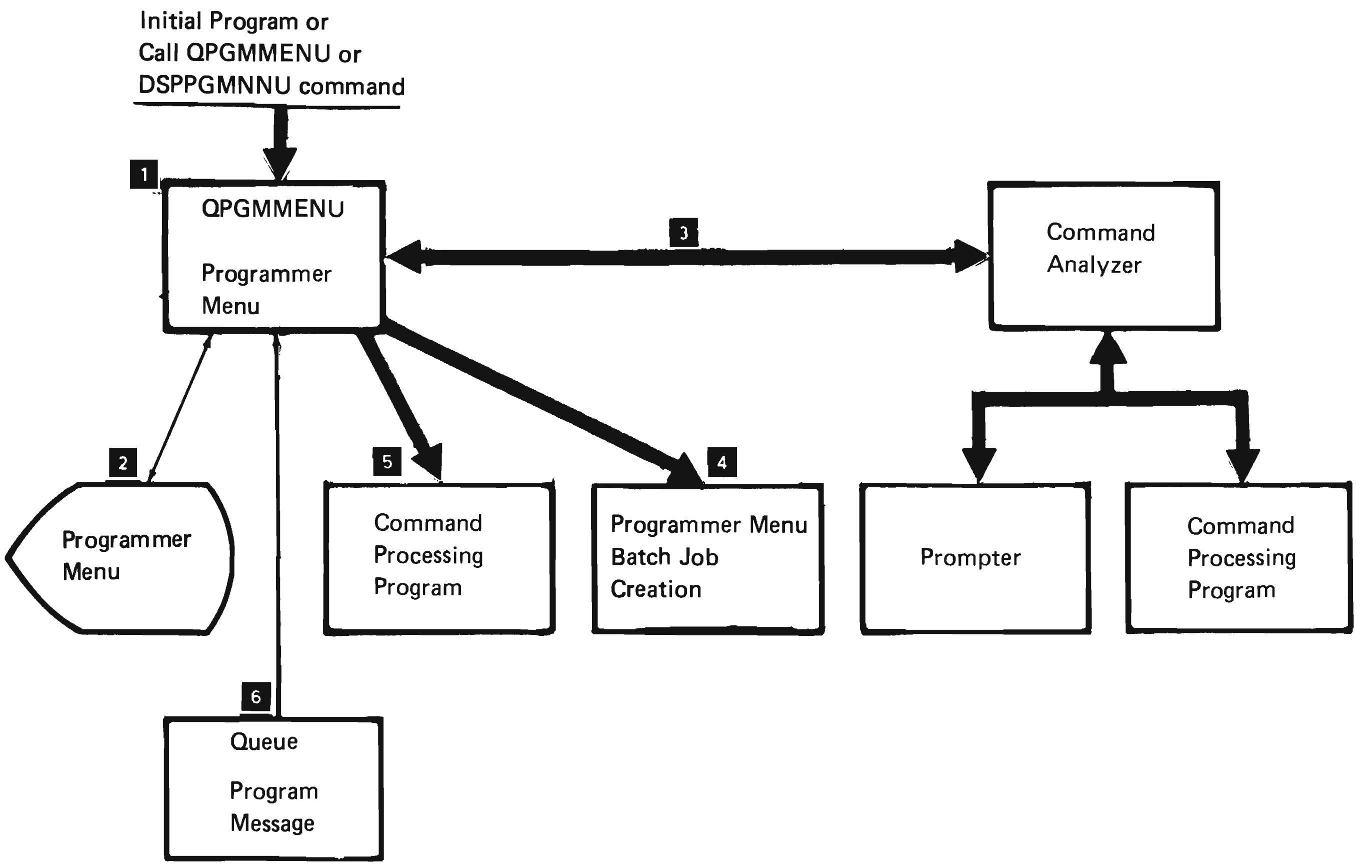


Figure MN-5. Programmer Menu Overview

System Request Menu Overview

Figure MN-6 and the following text describe the functions of the system request menu.

1 QMNSYSRQ is called by QWTPMSRQ when the system request key is pressed.

2 The system request menu is displayed. The user can select an option number and key in parameters. The user can bypass displaying the system request menu on a work station by keying in, on the input line at the bottom of the screen, an option and the parameters that are desired and/or needed. The system request menu is displayed again if no information was keyed in, if an invalid option number or incorrect parameters were keyed in, or if a message resulted when the option was processed. If no messages result from the requested operation when parameters are passed in, the system request menu is not displayed.

3 QMNSYSRQ calls the command analyzer and passes the command or command name to it, along with the prompt/no prompt and execute options, optionally calls the prompter to prompt for any required parameters that are missing, and then calls the appropriate command processing program to execute the command.

4 If executing the command causes an exception message to be sent, the exception message is displayed in the error message line at the bottom of the system request menu.

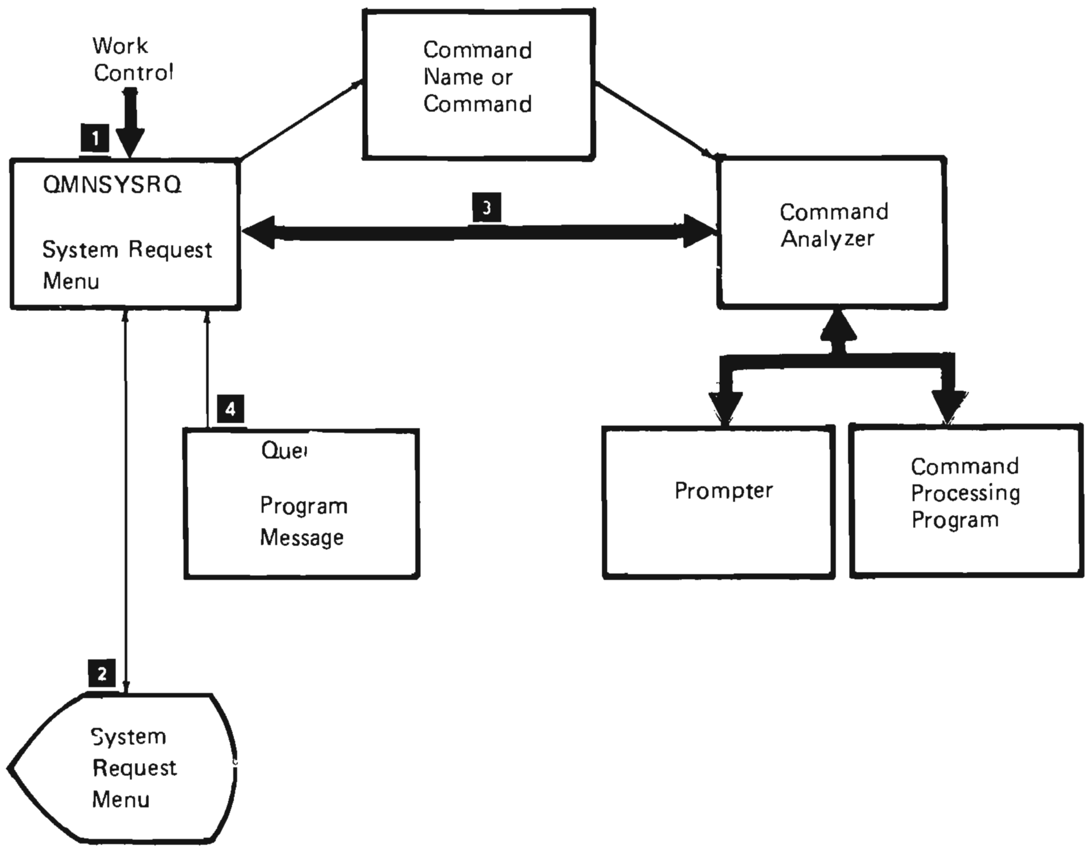


Figure MN-6. System Request Menu Overview

Build Menu Text Space Object Overview

Figure MN-7 and the following text describe the functions of the build menu text space object.

1 QMNTXTBL is called to build the menu index for the selection menus display and the menu space for the configuration menu displays.

2 QMNCMDTX is used to build the displays. It contains the names of the commands and command selection menus, formatting information, message IDs for the headings and menus, and optionally, message IDs for commands' prompt text.

3 If the device file contains a command name without any message IDs, the command definition object for that command is used to obtain the prompt text.

4 A message is retrieved from QCPFPMT for each message ID in the device file.

5 A listing is produced that contains the text received from the device file, the text for each menu line, and error information such as, command definition object not found, too many options specified, or prompt text truncated.

6 A space object is created that contains the menu text for either the command name prompt or configuration menu.

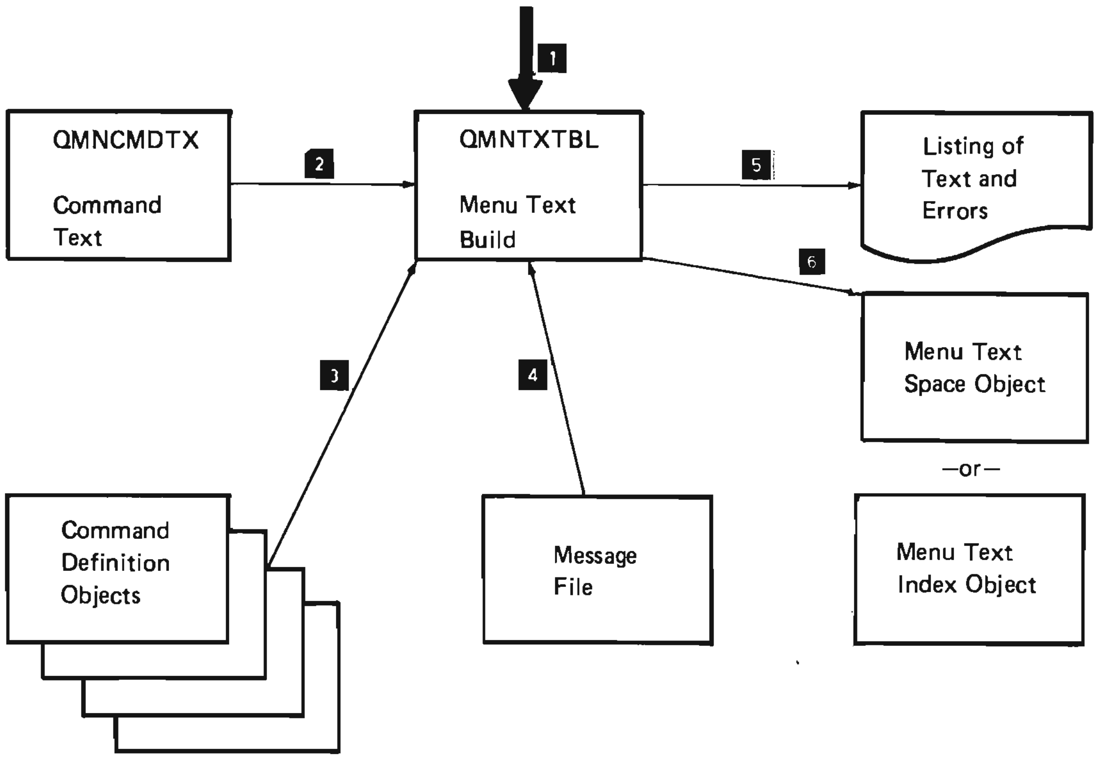


Figure MN-7. Build Menu Text Space Objects Overview

Network Facilities

INTRODUCTION

The Network Facilities (NF) component of the CPF is used to provide local and remote distributions of data files, save files, job streams, spooled files, and messages. The NF component relies on the system distribution directory provided by the Office Systems (OS) component to determine the destination of the distributions, and on the SNA Distribution Services (SNADS) component to perform the remote distributions.

Network File Queues

When the distribution of a network file arrives for a user, it is placed on the network file queue. The network file queue consists of entries on the distribution/recipient queue created by the directory management services of the OS component. Each entry on the network file queue points to a network file object, which is a space object containing the file data plus the control information.

Network Job Entry Table

The network job entry table is used to control the disposition of job streams sent to the system. It contains an entry for each user or group of users who may submit jobs to the system. It consists of a keyed physical data base file, QANFNJE, in library QUSRSYS.

GENERAL OVERVIEW

The NF component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows that it is dependent on a previous module.

Network Job Entry Management Modules

-->QNFJOBAU–Network Job Entry Management (ADDNETJOBE, CHGNETJOBE, RMVNETJOBE): This module processes the Add Network Job Entry, Change Network Job Entry, and Remove Network Job Entry commands.

-->QNFJOBDS–Display Network Job Entries (DSPNETJOBE): This module processes the Display Network Job Entries command.

Distribution Modules

-->QNFSNDTA–Send Network File (SBMNETJOB, SNDNETF): This module verifies that the request is valid, builds the necessary distribution control blocks, creates the network file object, and invokes the OS component distribution function to perform the distribution.

-->QNFSNSPL–Send Network Spooled File (SNDNETSPLF): This module verifies that the request is valid, builds the necessary distribution control blocks, invokes the copy spooled file function to copy the spooled file data into a data base file, creates the network file object, and invokes the OS component distribution function to perform the distribution.

-->QNFSNMSG–Send Network Message (SNDNETMSG): This module verifies that the request is valid, builds the necessary distribution control blocks, and invokes the OS component distribution function to perform the distribution.

The following modules are referenced by several other modules. More detail on these modules is shown in the command overviews later in this section.

QNFDSTRB–Perform Local Distribution: This module is invoked by the OS component distribution function once for each destination user. For local distributions, QNFDSTRB is invoked in the process in which the distribution is initiated. For remote distributions, it is invoked in the transaction program process that runs in the QSNADS subsystem on the remote system. Depending on the type of distribution, the module creates a copy of the network file object for the recipient, submits the job stream, writes the spooled file to an output queue, or sends the network message to the recipient's message queue. It also sends messages to notify the originator and the recipient of the arrival of the distribution.

QNFPACK–Create Network File Object: This module reads the file and creates the network file object, using the SNADS general file server write functions.

QNFUNPAK–Unpack Network File Object: This module is invoked to write the data from the network file object into the spooled output file.

Transaction Program Modules

-->QNFSBMTP–Submit Transaction Program: This module is invoked when the QSNADS subsystem is started. It submits a job to the QSNADS job queue to execute the transaction program, QNFTPDTA.

-->QNFTPDTA–Network Facilities Transaction Program: This module executes in the SNADS subsystem and processes incoming remote distributions received via SNADS.

Network File Processing Modules

-->QNFDSPRC–Display Network Files (DSNPETF): This module processes the Display Network File command.

-->QNFRCDTA–Receive Network Files (RCVNETF) and Cancel Network Files CNLNETF): This module processes the Receive Network Files command and the Cancel Network Files command.

-->QNFBROWS–Browse Physical File Member (BRWPFM): This module processes the Browse Physical File Member command. It is also invoked by QNFDSPRC to browse a network file object directly when the browse option is selected from the display.

Recovery Modules

-->QNFRBLDQ–Rebuild Recipient Queue Message: This module is invoked by the OS component module that rebuilds damaged recipient queues. It determines, based on the existence of network file objects, which messages must be enqueued on the new recipient queue for network files.

-->QNFRCLNF–Reclaim Network Files: This module is invoked during execution of the RCLSTG command. It ensures that all network file objects have a corresponding message on the correct recipient queue, and that all messages for network file objects on the recipient queues have corresponding network file objects. Damaged recipient queues are rebuilt, and damaged network file objects are destroyed.

This page is intentionally left blank.

Distribution Commands and Processing

Figure NF-1 and the following text describe the distribution function.

1 The SNDNETF and SBMNETJOB commands invoke QNFSNDTA to perform the distribution.

- **A** QNFSNDTA opens the file to be distributed, and invokes QNFPACK to create the network file object.
- **B** The OS general distribution module is invoked. If there are any remote distributions, this module invokes the SNADS distribute data function to perform the remote distribution **D**.
- **C** For each local distribution, QNFDSTRB is invoked. QNFDSTRB creates a copy of the network file object and updates the message to be put on the recipient queue. The general distribution module puts the message on the queue upon return from QNFDSTRB.

2 The SNDNETSPLF command invokes QNFSNSPL to perform the distribution.

- **A** QNFSNSPL invokes the spool component to copy the spooled file data into a data base file. QNFPACK is then invoked to create the network file object.
- **B** The OS general distribution module is invoked. If there are any remote distributions, this module invokes the SNADS distribute data function to perform the remote distribution **D**.
- **C** For each local distribution, QNFDSTRB is invoked. QNFDSTRB opens the spooled output file.
- **E** QNFUNPAK is invoked to write records from the network file object into the spooled output file.

3 The SNDNETMSG command invokes QNFSNMSG to perform the distribution.

- **B** The OS general distribution module is invoked. If there are any remote distributions, this module invokes the SNADS distribute data function to perform the remote distribution **D**.
- **C** For each local distribution, QNFDSTRB is invoked. QNFDSTRB sends the network message to the recipient's queue.

4 QNFTPDTA is invoked when the QSNADS subsystem is started, and until the subsystem is terminated. It invokes the SNADS receive distribution function, which waits for an incoming remote distribution. If the distribution is a status distribution, it sends a message to the recipient's message queue.

- **B** If the distribution is a data distribution, the OS general distribution module is invoked.
- **C** For each local distribution, QNFDSTRB is invoked. QNFDSTRB does one of the following, depending on the type of distribution:
  - For network files, creates a copy of network file object and updates the recipient queue message, which will be put on the queue by the general distribution module upon return.
  - For job streams, determines the action to be taken. If the job stream is to be filed, the action is the same as for network files. If the job stream is to be submitted, QNFPACK **E** is invoked to copy the data from the network file object into a data base file. Spooling is then invoked to submit the job stream to a job queue.
  - For spooled files, a spooled output file is opened. QNFPACK **E** is invoked to write records from the network file object into the spooled file.
  - For messages, the message is sent to the recipient's message queue.

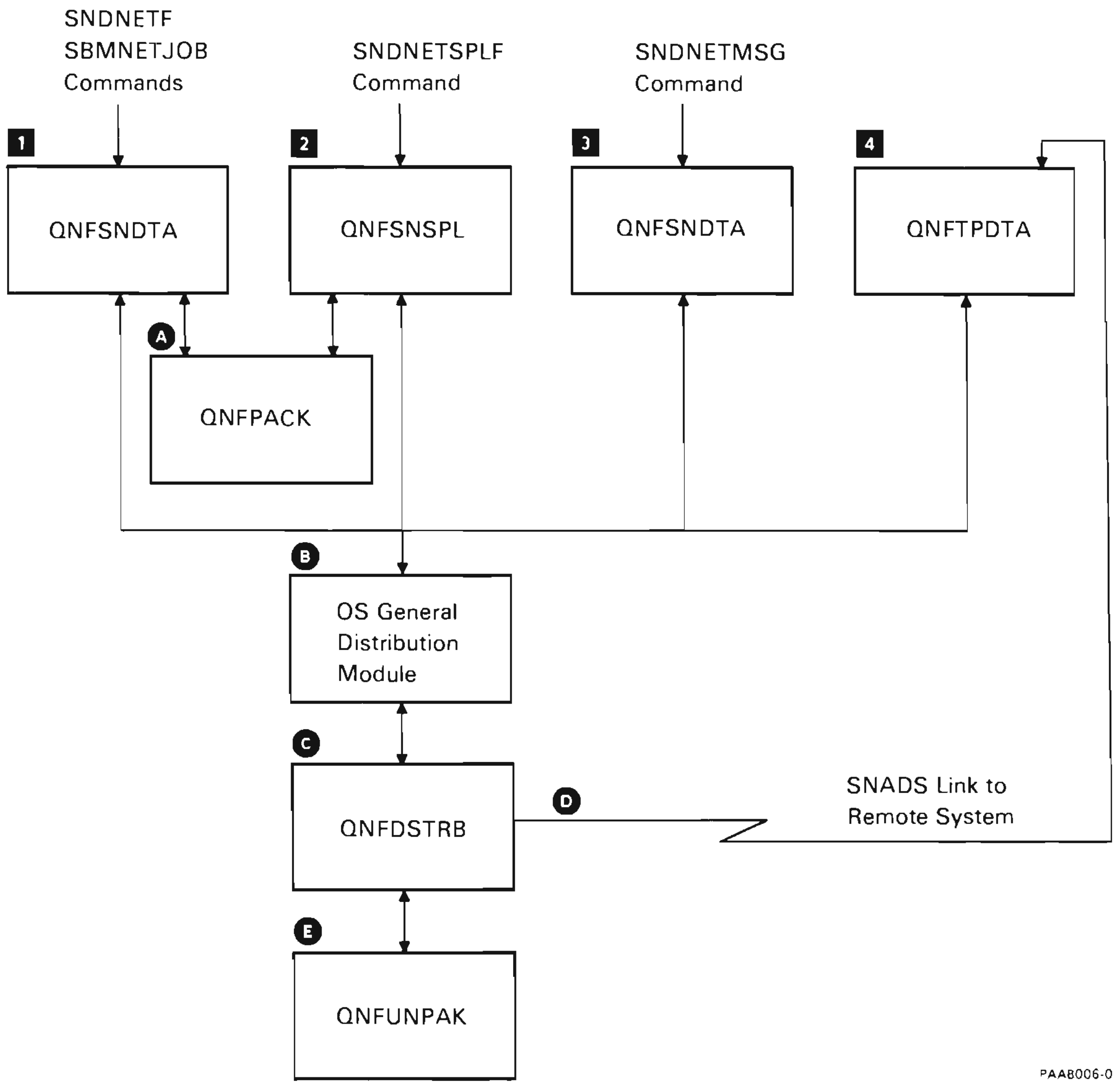


Figure NF-1. Distribution Commands and Processing

Figure NF-2 and the following text describe the operation of the Network File Processing commands, DSPNETF, RCVNETF, CNLNETF and BRWPFM.

1 The DSPNETF command invokes QNFDSPRC to perform the display function.

A If the RCVNETF or CNLNETF option is selected from the display, QNFRCDTA is invoked. Processing continues as described below for these commands.

B If the Browse option is selected from the display, QNFBROWS is invoked. QNFBROWS directly accesses the network file object and displays the data in the network file.

C If the Submit option is selected from the display, QNFUNPAK is invoked to copy the data from the network file object into a temporary data base file. Spooling is then invoked to submit the job stream to a job queue.

2 The RCVNETF and CNLNETF commands invoke QNFRCDTA to receive or cancel a network file.

E For the RCVNETF only, QNFUNPAK is invoked to copy the data from the network file object into the user-specified data base or save file. For both commands, the network file object is deleted and the corresponding message is removed from the recipient queue.

3 The BRWPFM command invokes QNFBROWS to display the contents of a data base file member.

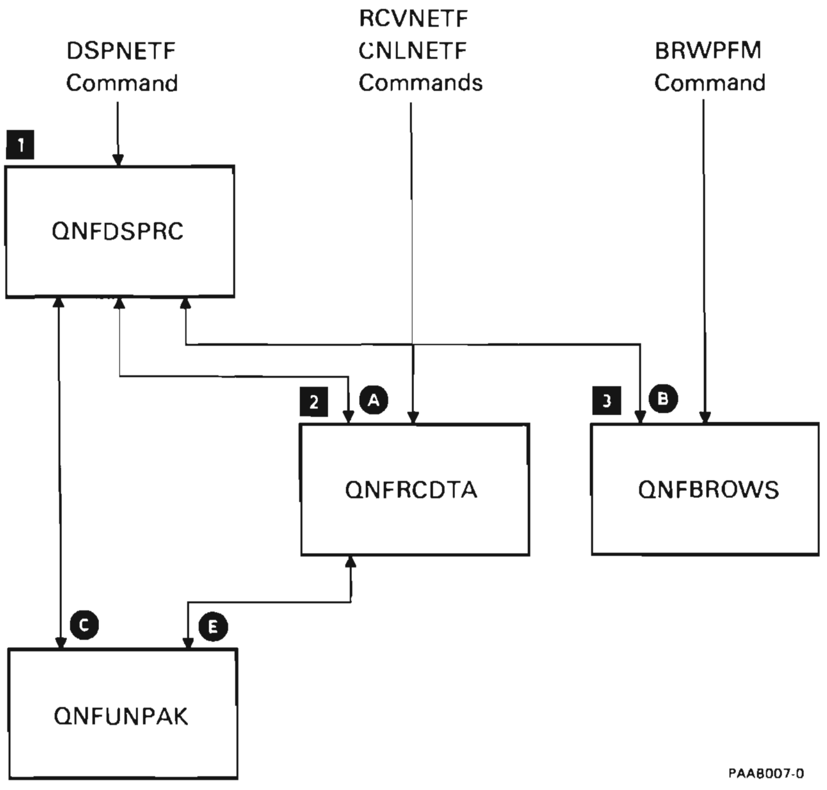


Figure NF-2. Network File Processing Commands

Figure NF-3 and the following text describe the operation of the ADDNETJOBE, CHGNETJOBE, RMVNETJOBE, and DSPNETJOB commands.

1 Module QNFJOBAU is invoked by the ADDNETJOBE, CHGNETJOBE, and RMVNETJOBE commands.

2 QNFJOBAU adds, deletes, and updates records in the Network Job Entry Table, which is in file QANFNJE.

3 Module QNFJOBDS is invoked by the DSPNETJOBE command.

4 QNFJOBDS reads records from the Network Job Entry Table, which is in file QANFNJE, and either displays or prints the records.

5 When the option to add, change, or remove a network job entry is selected from the display, module QNFJOBAU is invoked to perform the requested function.

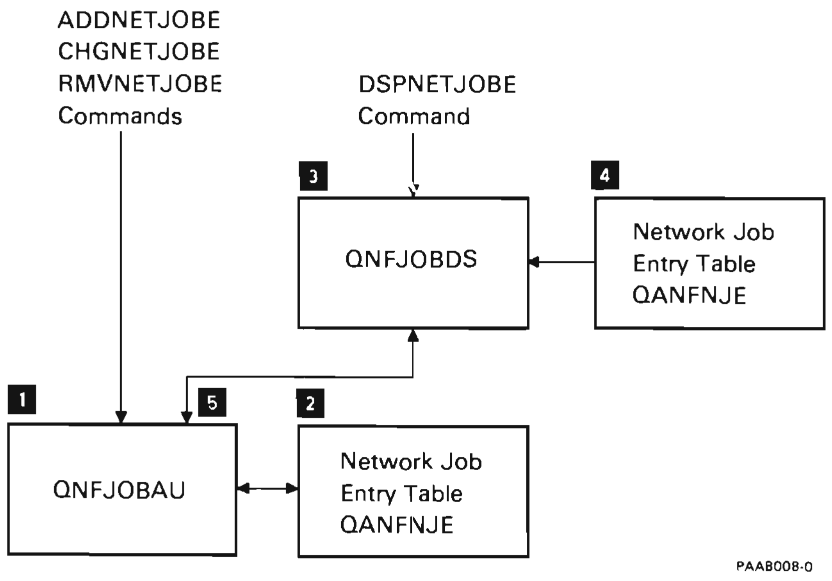


Figure NF-3. Network Job Commands

Figure NF-4 and the following text describe the processing performed to rebuild the messages on the recipient queue.

1. When a module detects a damaged recipient queue, it invokes QOSRCVRQ.

2. QOSRCVRQ creates a new recipient queue and performs the recovery for the OS component.

3. QNFRBLDQ is invoked to perform the recovery for the NF component.

4. QNFRBLDQ obtains a list of all network file objects for the recipient queue being recovered.

5. A message is put on the recipient queue for each network file object found for the recipient.

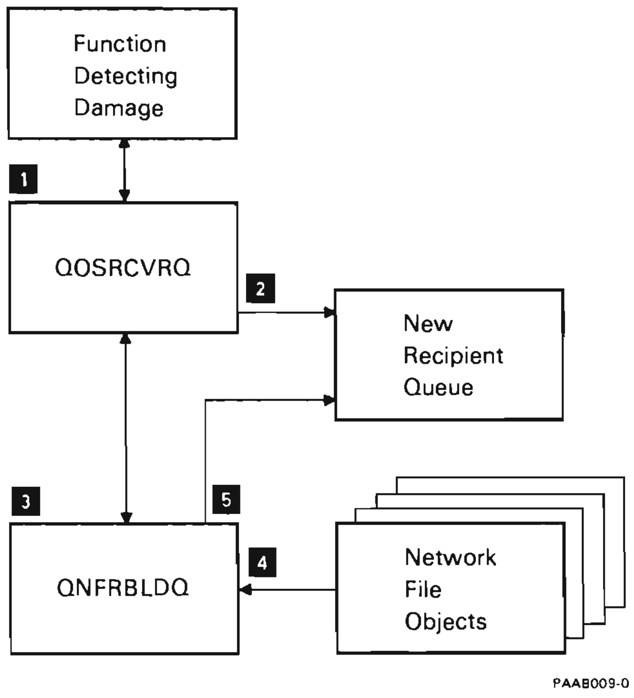


Figure NF-4. Rebuild Messages Processing

Figure NF-5 and the following text describe the operation of the network file recovery during reclaim storage processing.

1. During reclaim storage processing, QNFRCLNF is invoked by QRCLENUP.

2. QNFRCLNF obtains a list of all network file objects on the system.

3. QNFRCLNF ensures that there is a message on the recipient queue for each network file object. A message is put on the queue if necessary.

4. QNFRCLNF obtains a list of all recipient queues on the system. It checks each message on the queue to determine if a network file object exists for each message. If not, the message is removed from the queue.

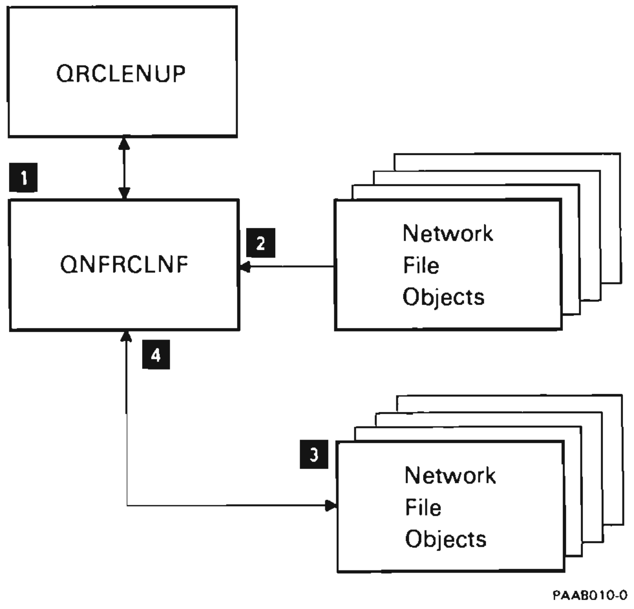


Figure NF-5. Network File Recovery

INTRODUCTION

The office systems component of the CPF provides support for the Document Interchange Architecture (DIA), provides an interface to the SNA Distribution Services (SNADS) support for cross system DIA services, and a variety of CL commands to set up, maintain, and manage an office network environment. Support is provided for the attachment of the IBM Display Writer system, the IBM Personal Computer, and other Office System Nodes such as System/36, other System/38s, the 5520 Administrative System, and the Distributed Office Support System (DISOSS) licensed program product. DISOSS executes on System/370, 43xx, and 30xx processors. In addition, the OS component provides these services to the OFFICE/38–Personal Services/38 licensed program product, which supports the 5250 and 3270 family of display terminals that can be attached to the System/38.

GENERAL OVERVIEW

The office system component functions in a variety of environments. CL commands can be used interactively or in batch programs to enroll, configure, authorize, and manage the system. The primary interface to these utilities is through the System/38 Command Analyzer. Another group of services is provided whereby an interactive session can be established with terminal nodes that support the Document Interchange Architecture. These services are provided for DIA commands sent to the System/38 by the terminal nodes. Devices may be attached through the Advanced Program-to-Program Communications (APPC) support (LU 6.2) or as emulating 5250 devices through the System/38 Host Router Support. In a network environment, office system component modules execute in jobs that run within the SNADS subsystem when DIA services are provided between various Office System Nodes.

Terminal Node Attachment

Figure OS-1 shows the module flow to support a terminal node such as the IBM DisplayWriter system or the Personal Computer when those devices request Document Interchange Services.

1 A terminal node sends an LU 6.2 Evoke to the System/38, which causes the QOSAPPC module to be started as the Problem Phase Program in the process initiated for the Evoke request. QOSAPPC opens a communications file to communicate with the device.

2 A terminal node attached as an emulating 5250 device sends a request to the Host Router to initiate a process for DIA services. The Host Router starts a process and provides a module that is the Problem Phase Program in the process. This module calls QOSSERVR to establish the environment for DIA.

3 QOSIEXIT is set up as the invocation exit for either QOSAPPC of QOSSERVR.

4 QOSINIT is called to create a space object, which contains the Office System Session Control Block and perform other process initialization.

5 Once the environment is initialized, QOSAPPC or QOSSERVR calls QOSCTRLR. QOSCTRLR is the Session Controller. Its function is to manage the flow of data in and out of the system and to manage the flow of control within the process.

6 If the attached device communicates using Advanced Program-to-Program Communications, QOSSIIO is called to send or receive data to/from the device. This module interacts with the Advanced Program-to-Program Communications Function Manager.

7 If the attached device communicates using 5250 emulation, QOSPCIO is called to send or receive data to/from the device. This module interacts with the System/38 Host Router.

8 As the Document Interchange Unit is received by the I/O modules, it is parsed by calling QOSPARSE to break the data data stream down into its constituent parts. When the entire DIU has been parsed, control returns to QOSCTRLR to proceed with the execution of the DIA command just received. (QOSPARSE may call QOSPRASP to initialize tables in the program associated space of QOSPARSE).

9 QOSIGNON is called to process a DIA Sign-On command and allow the session to continue. QOSIGNON uses information from the command to establish the session environment that will be used while communicating with the terminal node. It also changes the accounting code associated with this process to the accounting code of the user profile identified for the user who is signed on.

10 QOSSETCV is called to process a DIA Set Control Value command. This will provide a user with a document password, change an existing document password, or delete a user's document password.

11 QOSCHKAF is called by QOSCTRLR to perform temporary sign on processing of affinity processing. This is determined by the presence or absence of the password and source/recipient address operands on some of the DIA commands. QOSCHKAF calls QOSVFUSR to perform validation of the password (if present) and to validate that the user is enrolled in DIA services. QOSVFUSR may also lock the identified user's distribution/recipient queue to prevent its deletion while a DIA command is being processed.

12 Distribution Services commands are processed by DIA command processing programs shown in Figure OS-2.

13 Library Services commands are processed by DIA command processing shown in Figure OS-3.

14 The DIA session and the System/38 process is terminated when the DIA Sign Off command is received. This command is processed by QOSCTRLR.

15 If the user is enrolled in DIA services but the user's Distribution/Recipient Queue (DRQ) is damaged or destroyed, QOSRCVRQ is called to re-create the queue. All messages that were on the queue that point to distributions in progress are re-created and put on the queue.

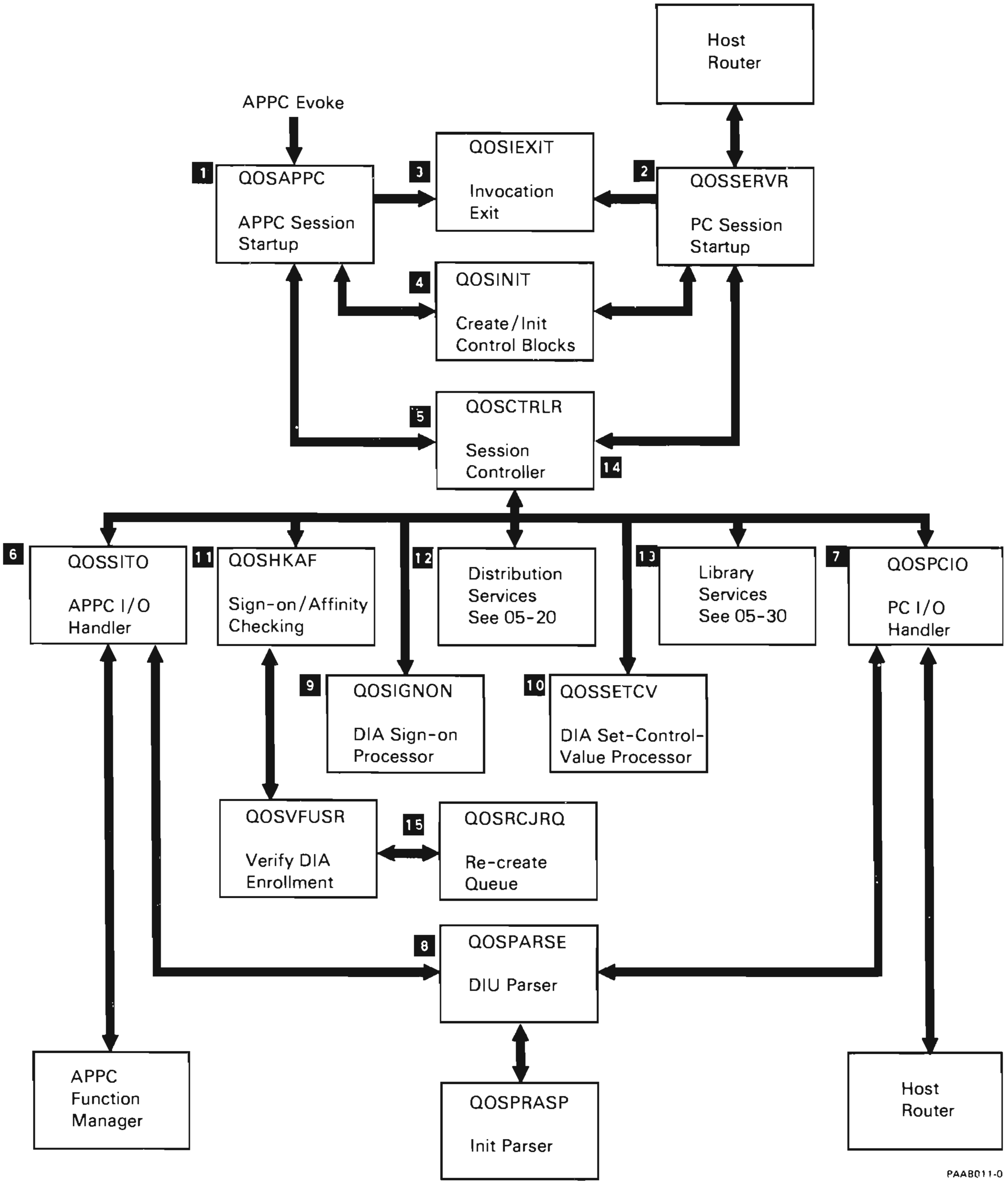

Host Router
APPC Evoke
1 QOSAPPC APPC Session Startup
3 QOSIEXIT Invocation Exit
2 QOSSERVR PC Session Startup
4 QOSINIT Create/Init Control Blocks
5 QOSCTRLR Session Controller
14
6 QOSSITO APPC I/O Handler
11 QOSHKAF Sign-on/Affinity Checking
12 Distribution Services See 05-20
13 Library Services See 05-30
7 QOSPCIO PC I/O Handler
9 QOSIGNON DIA Sign-on Processor
10 QOSSETCV DIA Set-Control-Value Processor
QOSVFUSR Verify DIA Enrollment
15 QOSRCJRQ Re-create Queue
8 QOSPARSE DIU Parser
APPC Function Manager
QOSPRASP Init Parser
Host Router
PAAB011-0

Figure OS-1. Terminal Node Attachment

Distribution Services Modules

Figure OS-2 is a continuation of Figure OS-1. It shows the support provided for Function Sets 2 and 5 of the Document Interchange Architecture—Distribution Services.

1 QOSOBTAN is called by the session controller to execute the DIA Obtain command. It processes any distributions waiting to be delivered to the recipient and delivers them to the terminal node by setting up the appropriate control blocks to allow the I/O modules to send the data to the Terminal Node.

2 QOSOBTAN serves as an interface with the SNADS component to send status back to the originator of a distribution that was sent with Confirmation-of-Delivery and the originator of the distribution was on a remote system.

3 QOSOBTAN calls QOSCKDLT once the distribution is complete to determine if the objects used in controlling the distribution can be deleted from the system. The distribution and the object used to manage the distribution may both be deleted.

4 QOSOBRCV is set up as the invocation exit for the QOSOBTAN module. It performs cleanup functions should the process abnormally terminate.

5 QOSLIST is called by the session controller to execute the DIA List command. It processes distribution information about items sent or waiting for delivery. It calls QOSLSTPT to build the formatted response to the List command. It calls QOSLSTUF to build the summary response to the List command.

6 QOSLIST calls QOSCKDLT once the status is complete to determine if the objects used in controlling the distribution can be deleted from the system. The object used to manage the distribution may be deleted.

7 QOSLSTIX is set up as the invocation exit for the QOSLIST module. It performs cleanup functions if the process abnormally terminates.

8 QOSREQDS is called by the session controller to execute the DIA Request Distribution command. QOSREQDS creates a permanent space object to save the document while it is being distributed. It calls QOSDSTRB to perform fan out of distribution lists, validate the recipients of the distribution, and complete the distribution function. QOSDSTRB places a message on the distribution/recipient queue of each local recipient. The message points to the document being distributed.

A QOSDSRCV is set up as the invocation exit for the QOSDSTRB module. It performs cleanup functions if the process abnormally terminates.

B QOSDSTRB calls QOSVFUSR to verify that each local recipient is enrolled in the system distribution directory.

9 QOSDSTRB serves as an interface with the SNADS component to send a distribution to a recipient on a remote Office System Node.

10 QOSRDRCV is set up as the invocation exit for the QOSREQDS module. It performs cleanup functions if the process abnormally terminates.

11 QOSREQDS calls QOSCKDLT if there are no valid recipients to determine if the objects used in controlling the distribution can be deleted from the system. The object used to manage the distribution may be deleted as well as the distribution data.

12 QOSCNLDS is called by the session controller to execute the DIA Cancel Distribution command. The module cancels either a distribution that is waiting to be delivered to the recipient or it cancels the tracking of a distribution that was sent to someone else.

13 QOSCNRCV is set up as the invocation exit for the QOSCNLDS module. It performs cleanup functions if the process abnormally terminates.

14 QOSCNLDS serves as an interface with the SNADS component to send status back to a remote recipient when a distribution was sent with Confirmation-of-Delivery and the originator of the distribution was on a remote system.

15 QOSRCVRQ is called when a user's Distribution/Recipient Queue is damaged or destroyed. It re-creates the queue and rebuilds messages to put back on the queue for distributions that are still in progress.

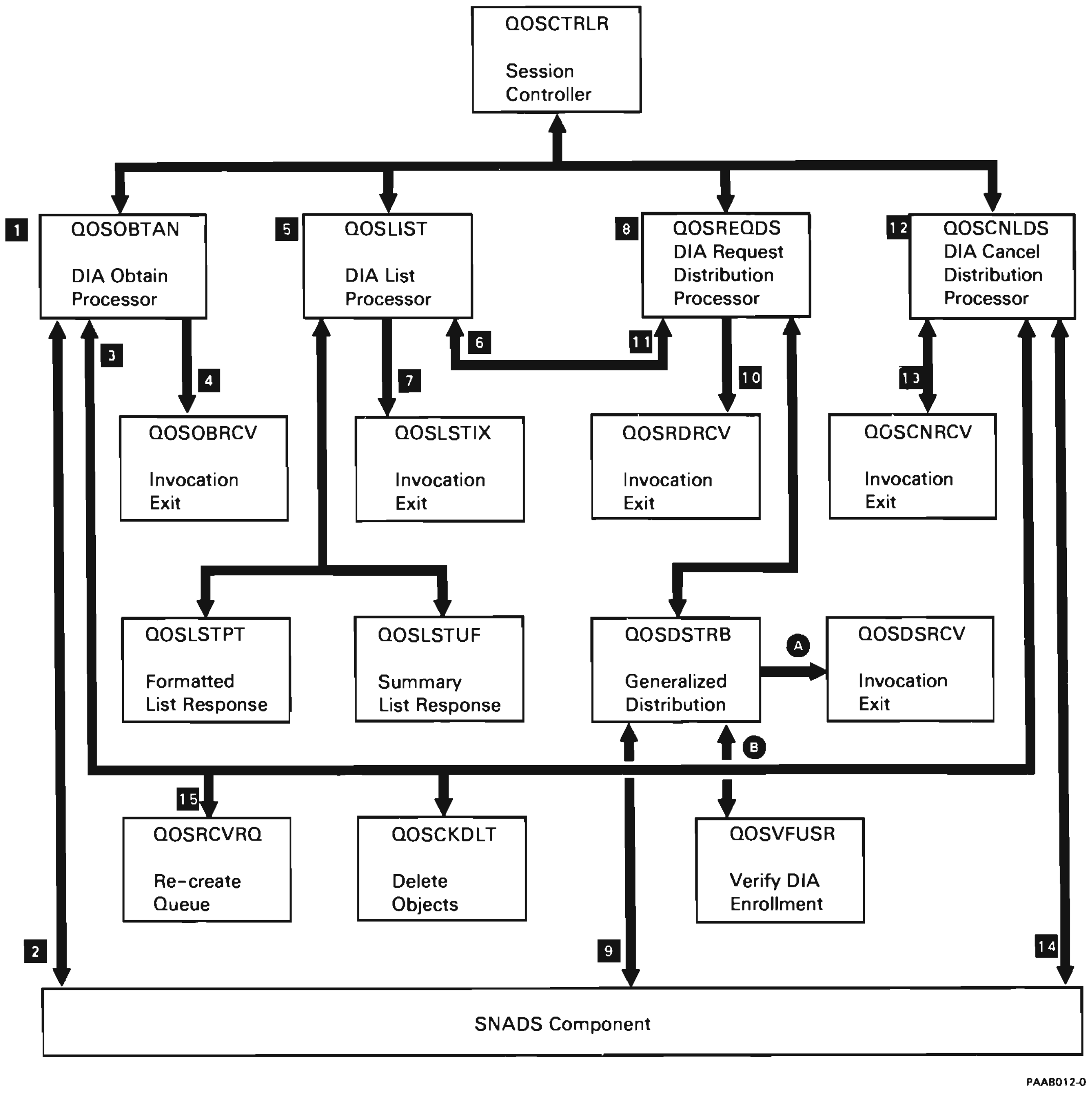


Figure OS-2. Distribution Services Modules

Library Services Modules

Figure OS-3 is a continuation of Figure OS-1. It shows the support provided for Function Set 8 of the Document Interchange Architecture–Library Services.

1 QOSFILE is called by the session controller to execute the DIA File command. It creates an object within the QDOC library and moves the document being sent by the terminal node to this object. It calls QOSIDPUP to perform document profile processing.

2 QOSIDPUP parses the Interchange Document Profile and adds searchable data to the System/38 data base files that make up the search index.

3 QOSFBKOT is set up as the invocation exit for the QOSFILE module. It performs cleanup functions if the process terminates abnormally. It is also called to perform cleanup functions when QOSFILE terminates normally.

4 QOSRETRV is called by the session controller to execute the DIA Retrieve command. Its function is to locate the named document, using the supplied Library Assigned Document Name or the Search Results List name and index, and deliver it to the terminal node.

5 QOSRETRV may call QOSCTDOC if the request is for a Document Descriptors Document. The requestor can name a document list and request that Interchange Document Profile parameters for documents within the list be returned. QOSCTDOC builds the response.

6 QOSRTRCV is set up as the invocation exit for the QOSRETRV module. It performs cleanup functions if the process abnormally terminates.

7 QOSEARCH is called by the session controller to execute the DIA Search Command. QOSEARCH builds a data base query request template and invokes the query support to determine which documents satisfy the selection criteria. Once the documents are selected, the Library Assigned Document Names (LADNs) are saved in a document list object. If the requestor does not want Interchange Document Profile parameters returned, QOSEARCH builds a response with a count of the number of documents selected.

8 QOSEARCH calls QOSCTDOC if Document Descriptors (IDP parameters) are to be returned as output from the search. QOSCTDOC builds the response called a document descriptors document.

9 QOSSBKOT is set up as the invocation exit for the QOSEARCH module. It performs cleanup functions if the process abnormally terminates. It is also called to perform cleanup functions when QOSEARCH terminates normally.

10 QOSTLMIT is a timer event handler that is set up if the requestor of Search puts a time limit on the search request. If time expires before the search completes, the event handler is invoked. It sends an escape message to QOSEARCH to let it know that it should terminate processing.

11 QOSDELET is called by the session controller to execute DIA Delete command. It locates the named document in the document library and deletes the requestor's ownership of the document.

12 QOSCKDLT is called by QOSDELET to determine if the document object can be deleted from the document library. If the primary owner and all secondary owners have deleted their ownership and the document and the document is not being distributed to anyone, the object is deleted.

13 QOSDLRCV is set up as the invocation exit for the QOSDELET module. It performs cleanup functions if the process abnormally terminates.

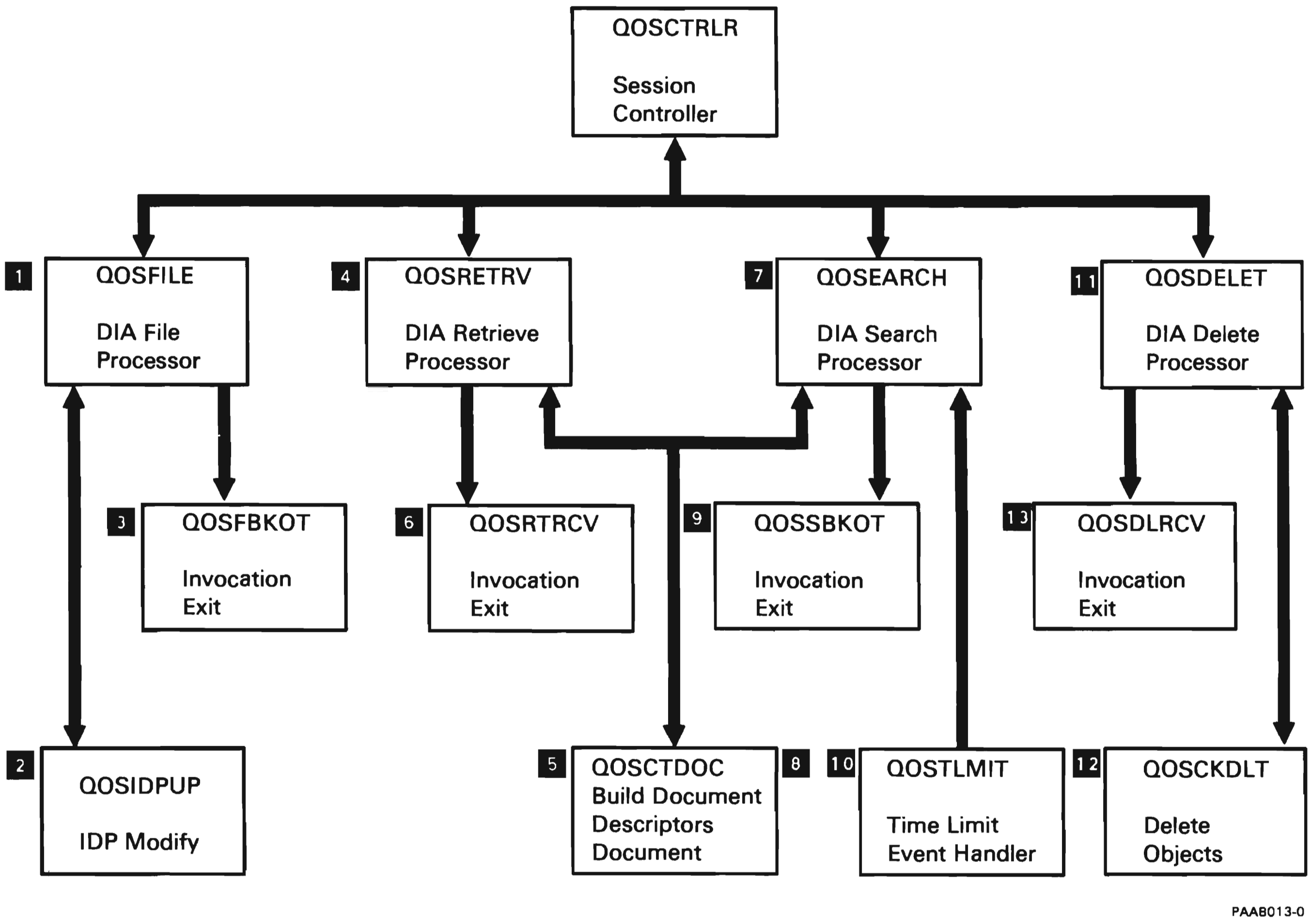


Figure OS-3. Library Services Modules

OFFICE/38–Personal Services/38 Interface Modules

Figure OS-4 shows the modules that present the Document Interchange interface of CPF to the OFFICE/38–Personal Services/38 licensed program product. These modules form a layer that is invoked through macros and map the requests to the interface used by the DIA command processing programs. To simplify the interface between the two System/38 products, a Document Interchange Unit (DIU) is not exchanged. Instead, macros are used that provide similar types of information as the DIU.

1 QOSMSCTL is called to establish a session environment between CPF and the OS/38 product. QOSMSCTL sets up control blocks and identifies the user who requests DIA services. It also is invoked when the session environment is to be terminated. QOSMSCTL simulates the support provided for the DIA Sign On and Sign Off commands.

A QOSVFUSR is called to verify that the person signing onto DIA for services is enrolled in the system distribution directory.

2 QOSINIT is called to create a space object that contains the Office Systems Session Control Block and perform other process initialization.

B QOSIEXIT is called to delete the Office Systems Session Control Block when the DIA session terminates.

3 QOSMLDOC is called for either a DIA Retrieve function or a DIA Delete function. This module determines which function is being requested by parameters passed from the calling module. It maps these parameters into the interface used by the DIA Delete or DIA Retrieve command processing program and transfers control to the appropriate command processing module to perform the requested function. Figure OS-3 describes the flow of control when QOSRETRV or QOSDELET are invoked.

4 QOSMDISP is called for a DIA File function, DIA Request Distribution function, or an Interchange Document Profile modify function. This module determines which function is being requested by the parameters passed from the calling module. It maps these parameters into the interface used by the DIA File or DIA Request Distribution or Modify command processing program and transfers control to the appropriate command processing module to perform the requested function. Figures OS-2 and OS-3 describe the flow of control when QOSFILE, QOSREQDS, or QOSIDPUP are invoked.

5 QOSMRECV is called for either a DIA Obtain function or a DIA Cancel Distribution function. This module determines which function is being requested by parameters passed from the calling module. It maps these parameters into the interface used by the DIA Obtain or DIA Cancel Distribution command processing program and transfers control to the appropriate command processing module to perform the requested function. Figures OS-2 and OS-3 describe the flow of control when QOSOBTAN or QOSCNLDS are invoked.

6 QOSMSRCH is called for the DIA Search function. It sets up the interface to QOSEARCH and invokes that module to start the search request. The interface between CPF and OFFICE/38–Personal Services/38 allows a search to return some of the information before the search actually completes. To continue the search request, OFFICE/38–Personal Services/38 invokes QOSMSRCH again. QOSMSRCH can complete returning the data without calling QOSEARCH.

7 QOSMLIST is called for the DIA List function. It sets up the interface to QOSLIST and invokes that module to perform the request. Figure OS-2 describes the flow of control when QOSLIST is invoked.

8 QOSCHKAF is called by the macro processors to verify that one person is authorized to work in place of another person.

9 QOSVFUSR is called to verify that an individual is enrolled in the system distribution directory.

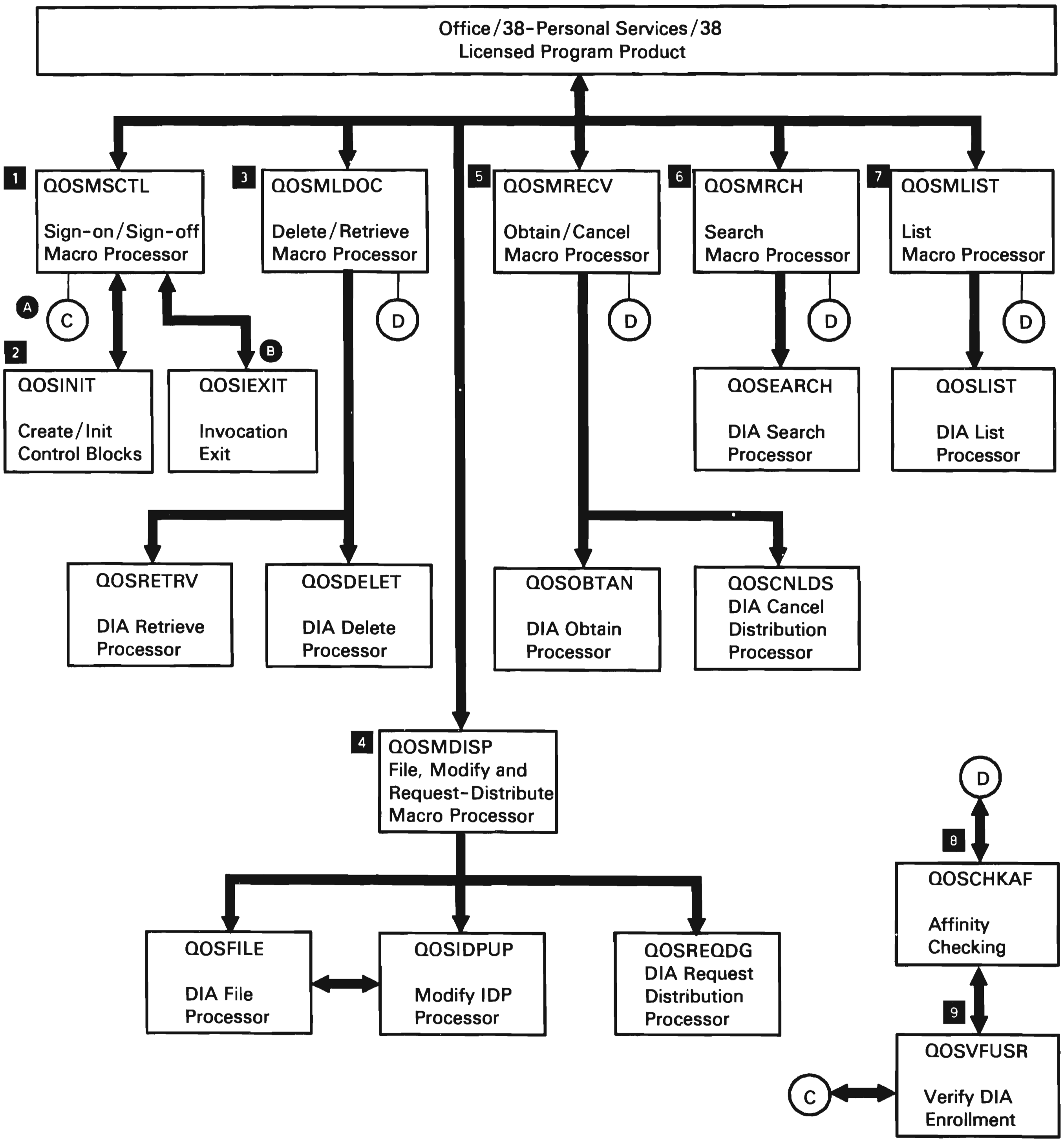


Figure OS-4. OFFICE/38–Personal Services/38

SNADS Subsystem Modules

Figure OS-5 shows the modules that execute within the SNADS subsystem in support of the Document Interchange services.

1 QOSSBMTR is called during the start up of the SNADS subsystem. It submits a job to the SNADS job queue to start execution of the DIA transaction program. The submitter module allows DIA dependent parameters to be set up independently of SNADS.

2 QOSDIATP runs in the SNADS subsystem as a never-ending job. It interfaces with the SNADS support to accept and process incoming distributions intended for DIA. It accepts documents and status information, then calls the appropriate module to process the data. The job is started when the SNADS subsystem starts and the job terminates when the SNADS subsystem is terminated.

- A QOSIEXIT is set up as the invocation exit for QOSDIATP. It performs cleanup functions if the process abnormally terminates.

- B QOSMSCTL is called to perform process initialization for the SNADS environment.

- C QOSINIT is called to create the Office Systems Session Control Block and perform some common initialization.

3 QOSRCVDC is called when a document has been received that must be delivered to users that are local to this system. Its function is to perform distribution of the document to each of the local recipients identified on the distribution.

4 QOSDSTRB is called by QOSRCVDC to expand distribution lists into their individual entries and to put messages on distribution/recipient queues that point to the document being distributed. If the expanded recipient list has remote users, SNADS is not called to send to these remote users. Expansion of distribution lists in the SNADS environment does not support remote users. Remote users are treated as invalid users.

- A QOSVFUSR is called to verify that local recipients are enrolled in the system distribution directory.

5 QOSRCVDC may serve as an interface to the SNADS support if some error status must be returned to the originator of the distribution.

6 QOSRCVST is called to process status being returned to the originator of the distribution. When a distribution is sent with Confirmation-of-Delivery or when errors are detected, status is returned to the originating system. This module updates the Distribution Tracking Object with the status.

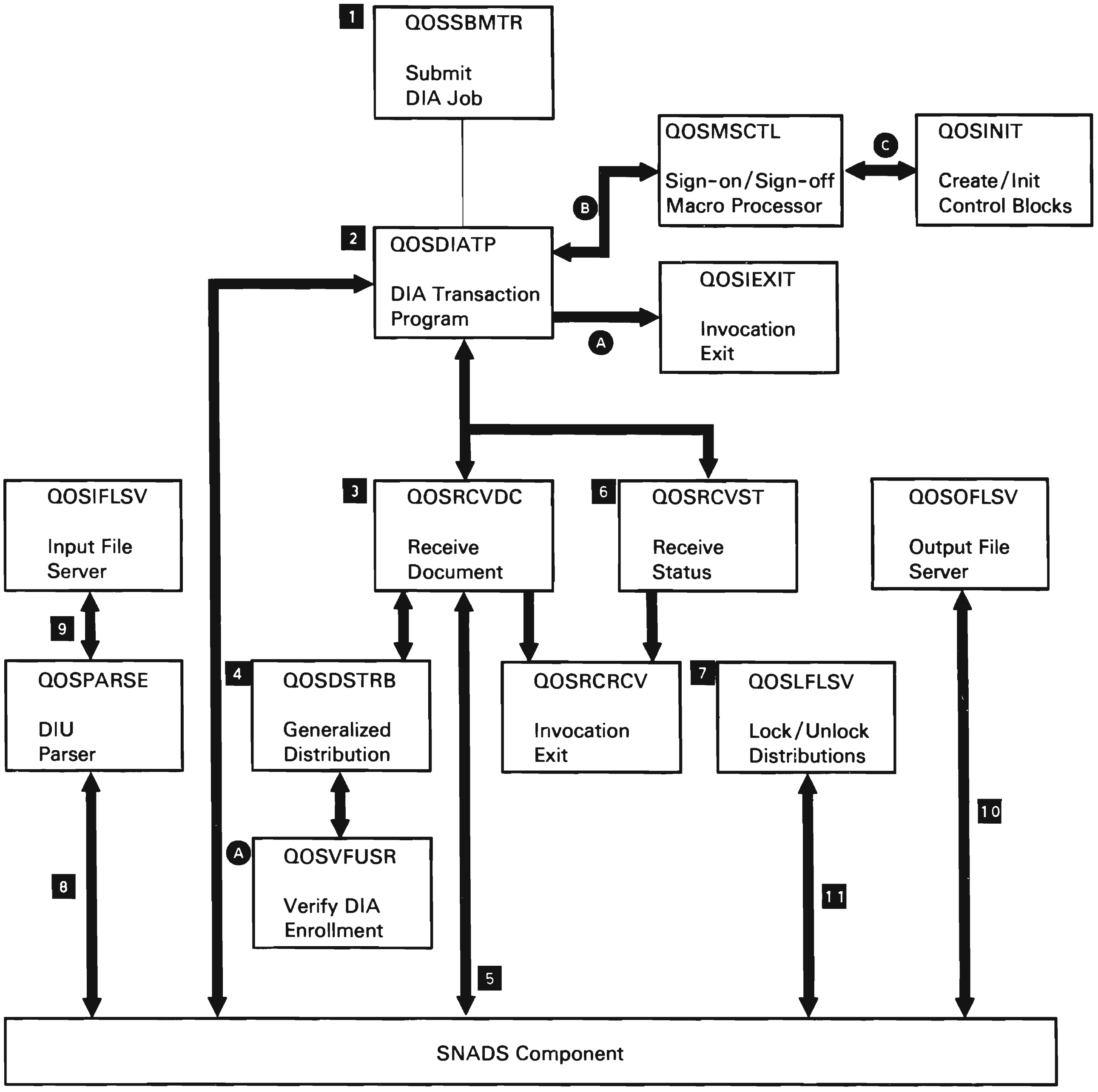
7 QOSRCRCV is set up as the invocation exit module for both QOSRCVDC and QOSRCVST. It will be invoked to perform cleanup if the process terminates abnormally.

8 QOSPARSE is invoked by the SNADS component to parse SNADS Distribution Interchange Units.

9 QOSIFLSV is invoked by QOSPARSE to handle incoming distributions. It creates an internal document object to store the distribution.

10 QOSOFLSV is invoked by the SNADS component when a DIA distribution is sent to another system. QOSOFLSV moves the data to be sent into the SNADS I/O buffer so that it can be transmitted.

11 QOSLFLSV is invoked by the SNADS component to lock and unlock document objects to manage them while they are in the process of being distributed. System/38 locks are not used but a usage count within the object serves as a logical lock.



PAA8015-0

Figure OS-5. SNADS Subsystem Modules

Command Language (CL) Command Processing Programs

Figure OS-6 shows the flow of control for CL commands.

Save Document Command (SAVDOC)

1 QOSSAVCP is called by the command analyzer for the SAVDOC command. It identifies and locks the documents to be saved.

2 QOSSRIXT is set up as the invocation exit for the QOSSAVCP module. It performs cleanup functions if the process abnormally terminates.

3 QOSOPEN is called to open data base files that contain the search index information associated with the document.

4 If a search is required to identify the documents to be saved, QOSSAVCP serves as an interface to the data base query component to determine which documents satisfy the selection criteria.

5 The Save/Restore component is called to save the document objects to tape or diskette. Once the documents are saved, control returns to QOSSAVCP. If STG(*DELETE) was specified on the command, the document and its data base search index information are deleted. If STG(*FREE) was specified on the command, storage for the document object is freed. All document object locks are released and a printed listing is produced if requested.

Restore Document Command (RSTDOC)

6 QOSRSTCP is called by the command analyzer for the RSTDOC command.

7 QOSSRIXT is set up as the invocation exit for the QOSRSTCP module. It performs cleanup functions if the process abnormally terminates.

8 The Save/Restore component is called to restore the document objects from tape or diskette into the QTEMP temporary library. When the documents are restored, control returns to QOSRSTCP where the documents are moved one by one from the QTEMP library to the QDOC library and search index information is updated to match the information associated with the restored document. If for some reason the document cannot be moved into QDOC, a diagnostic message is sent and the document is deleted from QTEMP.

Delete Document Command (DLTDOC)

9 QOSDLDCP is called by the command analyzer for the DLTDOC command. It verifies the validity of the request and locks the document objects to be deleted.

10 QOSMSCTL is called to set up DIA session control blocks that are used to perform an IPL recovery if the system fails in the middle of the delete document request.

11 If a search is required to identify the documents to be deleted, QOSDLDCP serves as an interface to the data base query component to determine which documents satisfy the selection criteria.

12 Once the documents have been identified, the documents are deleted from the QDOC library and any information residing in the data base search indexes is deleted.

Delete Document List Command (DLTDOCL)

13 QOSDLLCP is called by the command analyzer for the DLTDOCL command. It identifies the document list objects to be deleted and deletes them from the QUSRSYS library. In addition, it removes records from a cross-reference file for a data base name.

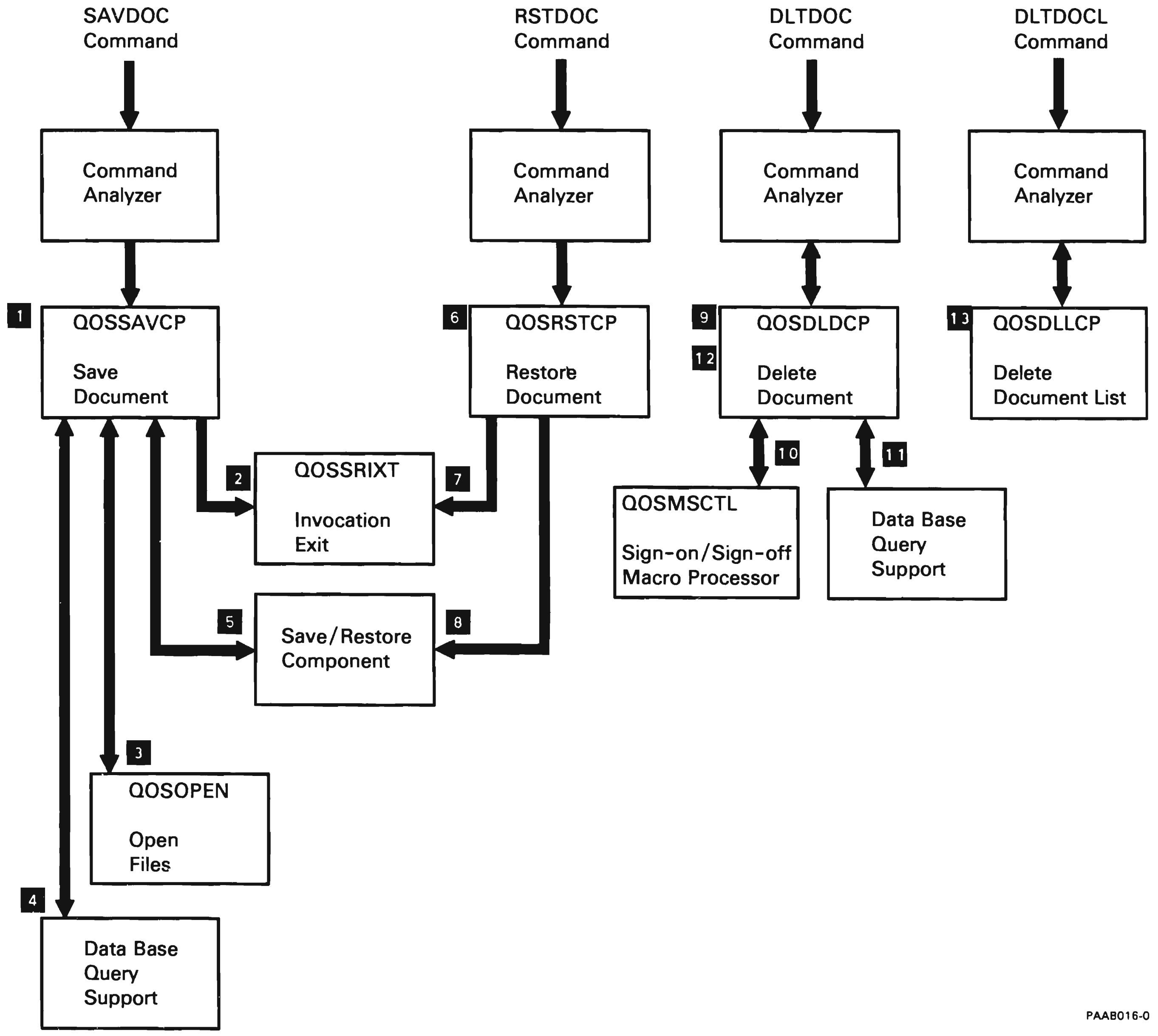


Figure OS-6. CL Command Processing Programs

Figure OS-7 shows the flow of control for the CL commands.

Grant Document Authority Command (GRTDOCAUT)

1 QOSGDACP is called by the command analyzer for the GRTDOCAUT command. It sets up the authorization for one person to work on behalf of another person. Both individuals must be enrolled in the system distribution directory. The authorization is maintained in data base records.

2 QOSVFUSR is called to verify that the users named on the command are enrolled in the system distribution directory.

Revoke Document Authority Command (RVKDOCAUT)

3 QOSSRDACP is called by the command analyzer for the RVKDOCAUT command. It revokes the authorization for one person to work on behalf of another user by deleting records from a data base file where the authorization is stored.

4 QOSVFUSR is called to verify that the users named on the command are enrolled in the system distribution directory.

Display Document Authority Command (DSPDOCAUT)

5 QOSDDACP is called by the command analyzer for the DSPDOCAUT command. It reads some data base files to find the information that indicates which users are authorized to work on behalf of other users and either formats a display or a printed listing with the information.

6 QOSVFUSR is called to verify that the users named on the command are enrolled in the system distribution directory.

Change Document Owner Command (CHGDOCOWN)

7 QOSCHGCP is called by the command analyzer for the CHGDOCOWN command. It updates two objects where DIA document ownership is stored. First, it updates the document object, then it updates data base files where the document search index information is stored.

8 QOSVFUSR is called to verify that the users named on the command are enrolled in the system distribution directory.

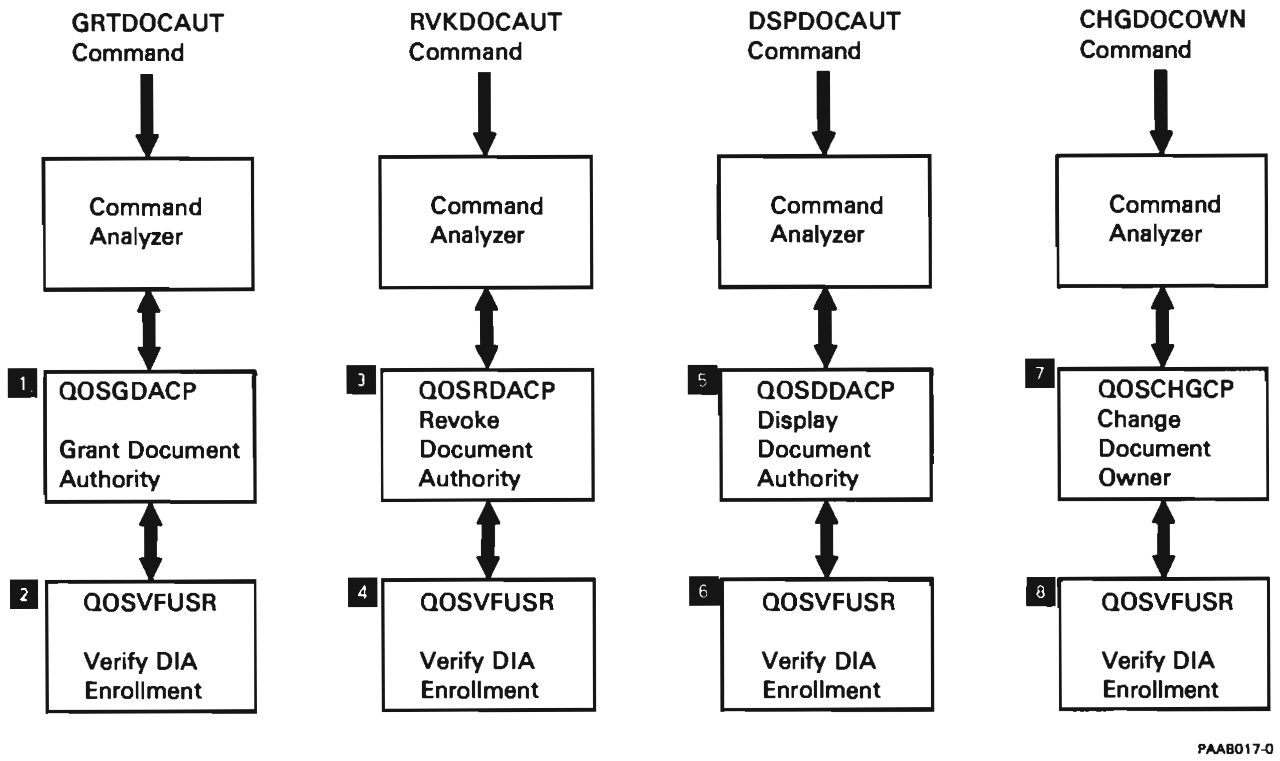

GRTDOCAUT Command
Command Analyzer
1 QOSGDACP Grant Document Authority
2 QOSVFUSR Verify DIA Enrollment
RVKDOCAUT Command
Command Analyzer
3 QOSRDACP Revoke Document Authority
4 QOSVFUSR Verify DIA Enrollment
DSPDOCAUT Command
Command Analyzer
5 QOSDDACP Display Document Authority
6 QOSVFUSR Verify DIA Enrollment
CHGDOCOWN Command
Command Analyzer
7 QOSCHGCP Change Document Owner
8 QOSVFUSR Verify DIA Enrollment
PAAB017-0

Figure OS-7. CL Command Processing Programs

Figure OS-8 shows the flow of control for the CL commands.

Grant Access Code Authority (GRTACCAUT)

1 QOSGAACP is called by the command analyzer for the GRTACCAUT command. It verifies that both the access codes and the users involved are defined on the system. It then authorizes use of the access codes by updating a data base record for each of the users being authorized to the access codes.

Revoke Access Code Authority Command (RVKACCAUT)

2 QOSRAACP is called by the command analyzer for the RVKACCAUT command. It verifies that the users involved are defined on the system. It then revokes use of the access codes by updating a data base record for each of the users whose authorization is being revoked.

Display Access Code Authority Command (DSPACCAUT)

3 QOSDAACP is called by the command analyzer for the DSPACCAUT command. It reads data from data base files to determine the access code each user is authorized to and formats the data on a display or a printed listing.

Add Access Code Command (ADDACC)

4 QOSADACP is called by the command analyzer for the ADDACC command. It determines that the access code is a valid number and that the access code is not currently defined on the system. It then adds the access code to a data base file.

Remove Access Code Command (RMVACC)

5 QOSRMACP is called by the command analyzer for the RMVACC command. It determines that the access code is a valid number and that the access code is currently defined in the system. It then removes the access code from all documents filed on the system that used this code, from all users who are authorized to this code, and finally from the data base files where the access code is defined.

A QOSRMIEX is set up for the invocation exit for QOSRMACP. It performs cleanup functions if the command terminates normally or abnormally.

Display Access Code Command (DSPACC)

6 QOSDATCP is called by the command analyzer for the DSPACC command. It reads a data base file where the definition of the access codes is stored and formats the information into a display or a printed listing.

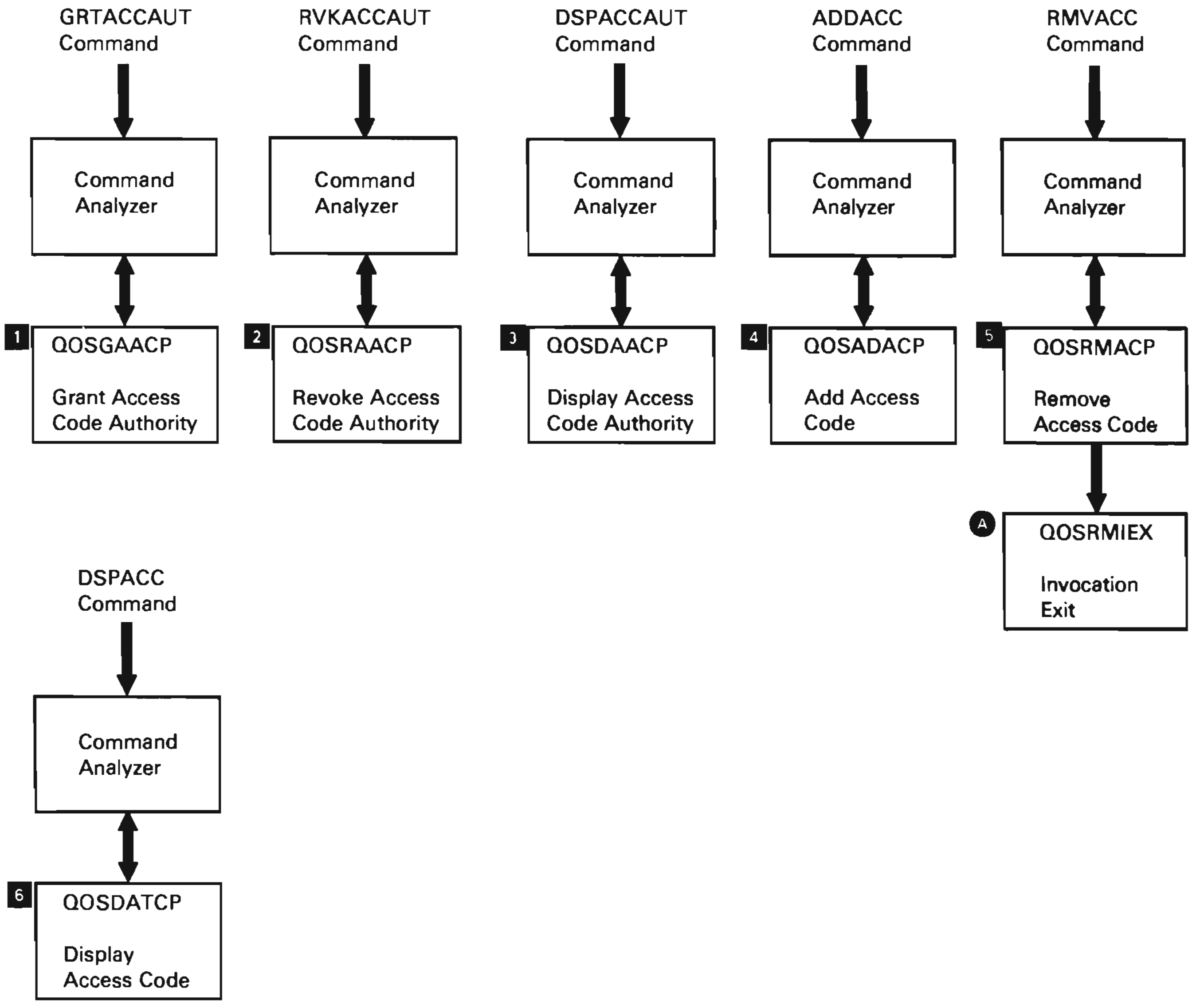

GRTACCAUT Command
RVKACCAUT Command
DSPACCAUT Command
ADDACC Command
RMVACC Command
Command Analyzer
1 QOSGAACP Grant Access Code Authority
2 QOSRAACP Revoke Access Code Authority
3 QOSDAACP Display Access Code Authority
4 QOSADACP Add Access Code
5 QOSRMACP Remove Access Code
A QOSRMIEX Invocation Exit
DSPACC Command
Command Analyzer
6 QOSDATCP Display Access Code
PAA8018-0

Figure OS-8. CL Command Processing Programs

Manage Directory Command (MNGDIR)

1 QOSENTCP is called by the command analyzer for the MNGDIR command.

2 QOSDIRSM is called from QOSENTCP when a user requests to view all the entries within the system distribution directory.

3 QOSRTENT is called to build the subfile that contains all the entries within the system distribution directory.

4 QOSDLUSR is called from QOSDIRSM when the user wishes to delete an entry from the system distribution directory.

5 QOSVFUSR is called from QOSENTCP to verify that a user is enrolled in the system distribution directory.

6 QOSDIRDL is called from QOSENTCP when a user's name is entered on the command. It produces a display of the detailed information about an entry within the system distribution directory.

7 QOSDIRDL is called from QOSDIRSM when the user wishes to view the detailed information about an entry within the system distribution directory.

8 QOSVFUSR is called from QOSDIRDL to verify that the user profile associated with a new entry being added is defined on the system.

9 QOSDLUSR is called from QOSDIRDL when the user wishes to change the user ID associated with an entry or when a local user is changed to a remote user. Under these conditions, a user ID may have to be deleted from the system to keep the data base files synchronized.

10 QOSVFUSR is called from QOSDLUSR to place an exclusive lock on the user's distribution/recipient queue in order to lock out all other operations while the user is being deleted.

11 QOSUSRDP is called from QOSDIRDL when the user wants to view all the user profiles on the system.

12 QOSSYSDP is called from QOSDIRDL when the user wants to view all the node IDs that have been defined on the system.

13 QOSCRTRQ is called from QOSDIRDL when a new local user is being enrolled in the system distribution directory. It creates an internal queue object that is used to manage distributions as they are in progress.

14 QOSDIEXT is set up as the invocation exit for the QOSENTCP module. It performs cleanup functions when the command terminates either normally or abnormally.

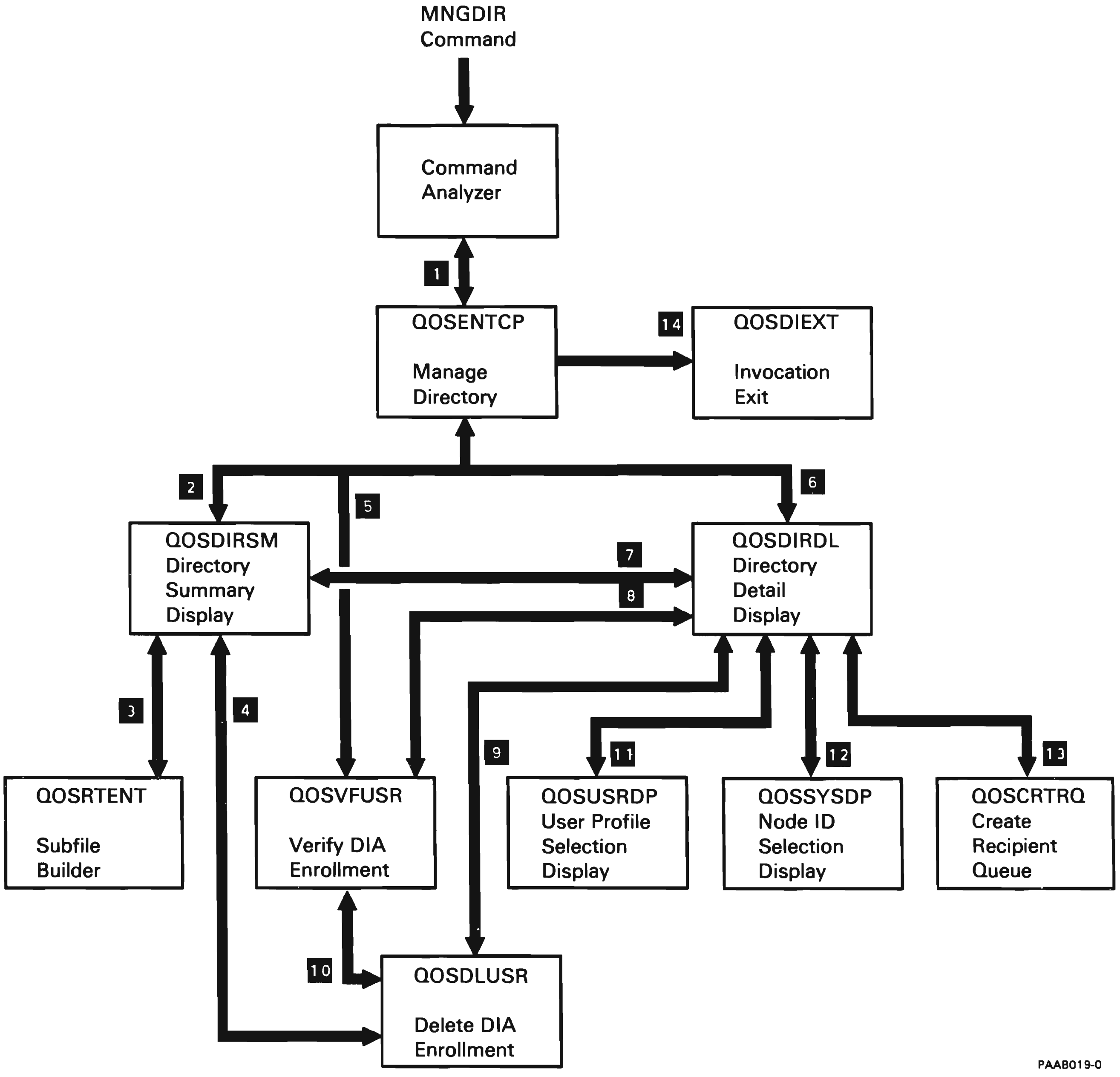


Figure OS-9. Manage Directory Command

Diaplay Directory Command (DSPDIR)

1. QOSDSDCP is called by the command analyzer for the DSPDIR command. It determines which function was requested by the user, opens the appropriate data base files, and calls the next module to perform the request.

2. QOSDIEXT is set up as the invocation exit for the QOSDSLCP module. It performs cleanup functions when the command terminates either normally or abnormally.

3. QOSDIRSM is called from QOSDSDCP when the request is to show all the users enrolled in the system distribution directory.

4. QOSRTENT is called to build the subfile that contains all the users enrolled in the system distribution directory.

5. QOSVFUSR is called from QOSDSDCP to determine if a user is enrolled in the system distribution directory.

6. QOSDIRDL is called from QOSDSDCP when a user wants to view the details of a particular entry within the system distribution directory. The user's name was entered in the command.

7. QOSDIRDL is called from QOSDIRM when a user wants to view the details of a particular entry within the system distribution directory. The user is selected from the list of users in the directory.

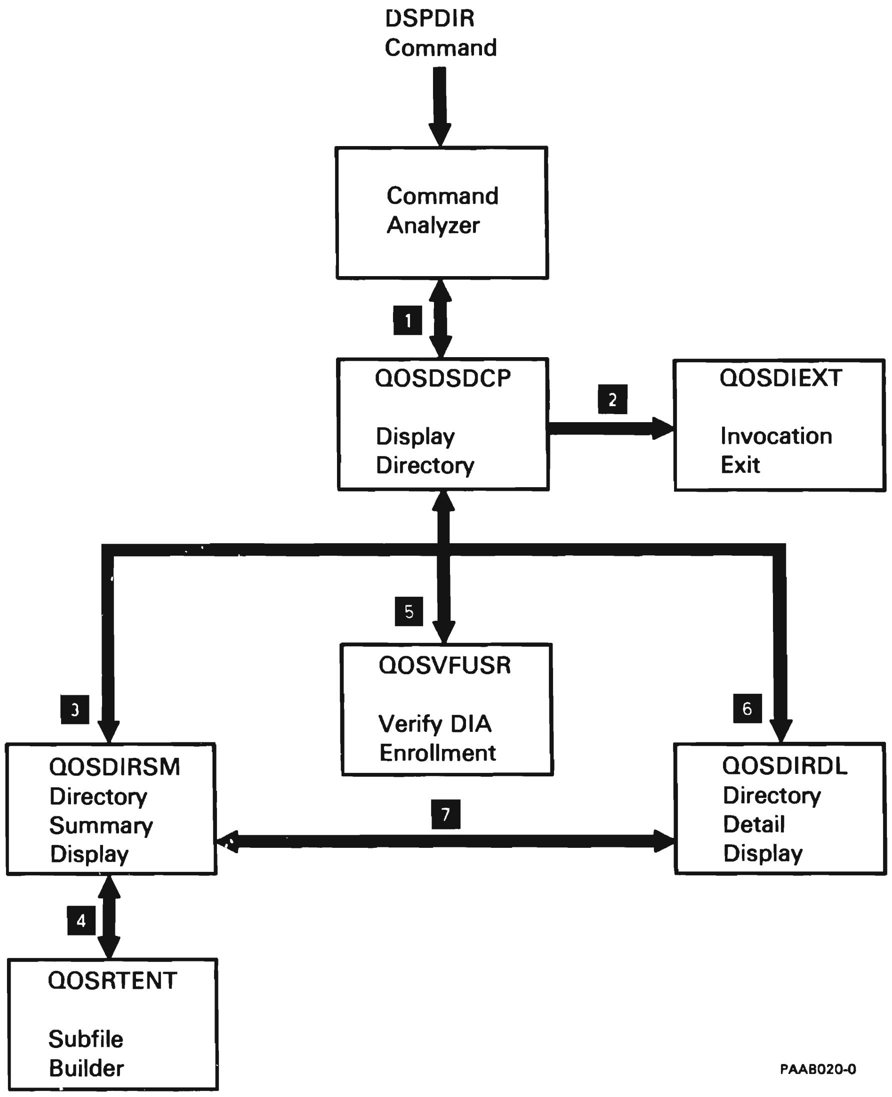

DSPDIR Command
Command Analyzer
1
QOSDSDCP Display Directory
2
QOSDIEXT Invocation Exit
5
QOSVFUSR Verify DIA Enrollment
3
6
QOSDIRSM Directory Summary Display
7
QOSDIRDL Directory Detail Display
4
QOSRTENT Subfile Builder
PAAB020-0

Figure OS-10. Display Directory Command

Manage Distribution List Command (MNGDSTL)

1 QOSDSLCP is called by the command analyzer for the ANGDSTL command. It determines which function was requested by the user, opens the appropriate data base files, and calls the next module to perform the request.

2 QOSLSTSM is called by QOSDSLCP when a list of all distribution lists currently defined on the system is requested. It produces the Manage Distribution Lists Display.

3 QOSRTENT is a common function module that builds subfiles. In this case, it is called by QOSLSTSM to build the subfile for the Manage Distribution Lists display.

4 QOSLSTDL is called from QOSLSTSM when a user wants to view the details of a distribution list. It displays the members of a distribution list.

5 QOSLSTDL is called from QOSDSLCP when the name of a distribution list is entered on the MNGDSTL command. QOSLSTDL produces a display showing the members of the list.

6 QOSRTENT is called from QOSLSTDL to build the subfile that contains the members of the distribution list.

7 QOSLSTDP is called from QOSLSTDL when a user wants to use another distribution list as a base for building a new distribution list or when an existing list is to be added to the current list being worked on.

8 QOSDIRDL is called from QOSLSTDL when a user wants to view the details of a system distribution directory entry.

9 QOSDIRDL is called from QOSSELDP when a user wants to view the details of a system distribution directory entry.

10 QOSSELDP is called from QOSLSTDL when a user wants to use a list of system distribution directory users to build a new distribution list or to add users to an existing list.

11 QOSLSTDL is called from QOSLSTDP when a user wants to view the details of a distribution list.

12 QOSRTENT is called from QOSLSTDP to build the subfile that contains all the distribution lists on the system.

13 QOSRTENT is called from QOSSELDP to build the subfile that contains all the users enrolled in the system distribution directory.

14 QOSDIEXT is set up as the invocation exit for the QOSDSLCP module. It performs cleanup functions when the command terminates either normally or abnormally.

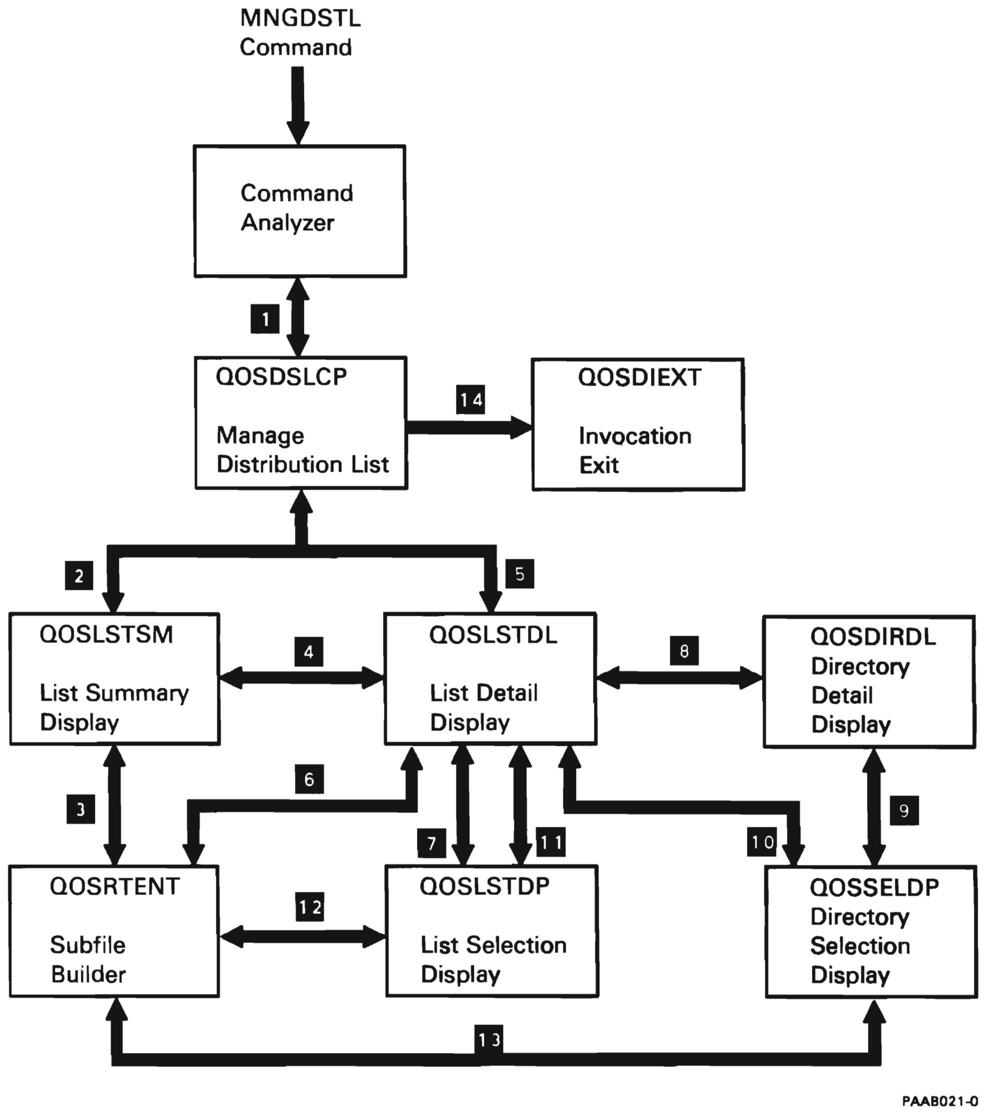

MNGDSTL Command
Command Analyzer
1
QOSDSLCP Manage Distribution List
14
QOSDIEXT Invocation Exit
2
5
QOSLSTSM List Summary Display
4
QOSLSTDL List Detail Display
8
QOSDIRDL Directory Detail Display
6
3
9
7
11
10
QOSRTENT Subfile Builder
12
QOSLSTDP List Selection Display
QOSSELDP Directory Selection Display
13
PAAB021-0

Figure OS-11. Manage Distribution List Command

5211/3262/3203 Function Manager

INTRODUCTION

The 5211/3262/3203 function manager component of the CPF (control program facility) provides the support for the 5211/3262/3203 Printer on System/38.

The following printer functions are supported by the 5211/3262/3203 function manager:

- Open printer file for processing
- Close printer file for processing
- Write data to a printer file

GENERAL OVERVIEW

5211/3262/3203 Function Manager Modules

The 5211/3262/3203 function manager consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry into the component.

-->QPNOPEN–Printer Open: This module prepares an output file for processing by a 5211/3262/3203 Printer. The printer is initialized and the LUD (logical unit description) is modified if any of the following are changed:
- Print image
- Forms length
- Translate table name
- Forms width
- Lines per inch to print

If the print belt is to be changed, a message is sent to the default message queue to change it. Open modifies the common data management entry point table, when field level support is specified, entering the address for QPNPTFLD in place of QPNPUT.

When the spool writer is printing data from the spool output queue, open modifies the common data management entry point table, entering the address of QPNREQIO instead of QPNPUT.

QPNALLOC–Continuation of Open: This module is part of the open process and performs those functions common to open processing for printer files. It is called by the 5211/3262/3203 function manager open, the 5224/5225/5256 function manager open, and the spool open to validate the open parameters and establish the function manager work area.

QPNOERRS–Error Handler: This module is called by QPNALLOC, QPNOPEN, or QWPOPEN when an open parameter error occurs.

-->QPNCLOSE–Printer Close: This module closes a file to the 5211/3262/3203 Printer. Blocked records are printed if the close is normal; records are purged if the close is not normal. If the close is not temporary, the space objects are destroyed.

-->QPNPUT–Put Records: This module places a single data record into a 5211/3262/3203 printer output file. Page formatting can be controlled by QPNPUT, by the user program, or from information in the device file depending on whether data records are described in the user program, outside the user program in a device file, or in both places. When print records are folded or truncated, a message is sent to the program and the job log indicating that occurrence.

-->QPNFEOD–Forced-End-of-Data: This module causes the printer function manager to print all data that has been blocked in the data buffer but not yet printed.

-->QPNEVT–Event Handler: This module handles the operator intervention required event.

QPNLUDIN–LUD-Associated Space Initialization: This module initializes the device-dependent area of the LUD.

-->QPNPTFLD–Put Records: This module works the same as QPNPUT but is used when field level support is specified. This module formats a single line, field by field, according to the specifications in the device file and has the capability of editing those fields.

QPNPERRS–Error Handler: This module is invoked by the put modules or spool intercept modules when a put parameter error occurs.

QPNXTANZ–Error Handler/Forms Alignment: This module handles exception conditions and hardware I/O errors as well as forms alignment.

-->QPNREQIO–REQIO Processor: This module is the interface to the IOM. It issues the REQIO instruction and its related processing for the put modules, and interfaces directly with the spool writer when a put operation is performed.

5211/3262/3203 Print Operation

Figure PN-1 and the following text show an overview of a 5211/3262/3203 print operation.

1 A high-level language program or the spooling component, through the QDMCOPEN module of common data management, calls QPNOPEN to prepare a file and, if necessary, to initialize the printer for a print operation.

A An argument list is passed that contains a pointer to the UFCB (user file control block) and an index into the ODPCB (open data path control block) for the device-dependent open.

B A message is sent to the default message queue if a different print belt is to be put on the printer.

If the lines per inch, print image, translate table, or forms length is changed from the previous file, an MODLUD is issued to the I/O manager.

2 After the file has been opened and the printer initialized, the information to be printed can be sent to the printer. This is done by a high-level language program or the spooling component invoking QPNPUT. Page formatting information can be found either in a device file or the user program.

A An argument list is passed that contains pointers to the UFCB, option list, and to control information.

B If an error is detected or the forms need to be positioned at line 1, a message is sent to the default message queue. When print records are folded or truncated, a message is sent to the job log indicating that occurrence.

C Request I/Os are issued to the I/O manager to print the records. Up to an entire page of print lines can be loaded into the data buffer before a print operation is performed.

When the spool writer is putting data to the printer, data blocks of 512 or 4096 bytes in SNA (systems network architecture) character stream format (data including control characters) are sent to the function manager.

3 QPNFEOD is called to perform a print operation on print lines that have been blocked in the data buffer but not yet printed.

A An argument list is passed that contains a pointer to the UFCB.

B If an error is detected or the forms need to be positioned at line 1, a message is sent to the default message queue.

4 An event is signaled by the I/O manager if the printer Stop/Reset switch is pressed.

B QPNEVT causes an intervention-required message to be sent to the default message queue. After the appropriate action has been taken by the default message queue and the Ready switch on the printer has been pressed, printing is resumed.

5 After all print records have been passed to the 5211/3262/3203 function manager, QPNCLOSE, through QDMCLOSE of data management, is called to close the file to further processing.

A An argument list is passed that contains a pointer to the UFCB, the type of close to perform, and an index into the ODPCB for the device-dependent close.

B If an error is detected or the forms need to be positioned at line 1, a message is sent to the default message queue.

C A print operation is performed to print those lines that have been blocked in the data buffer but not yet printed.

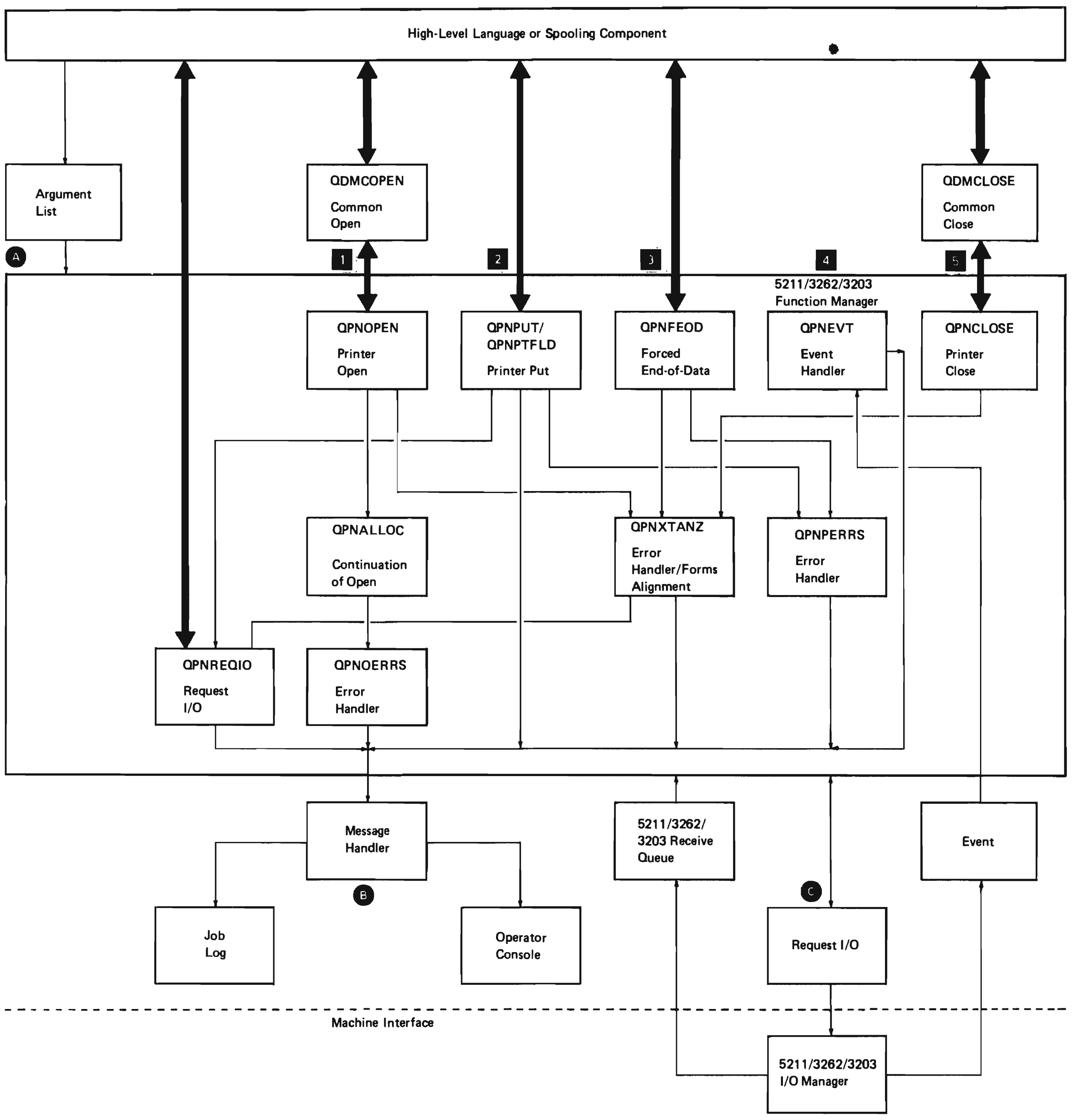


Figure PN-1. 5211/3262/3203 Print Operation Overview

INTRODUCTION

The PRM (program resolution monitor) component of the CPF (control program facility) converts programs in the IRP (intermediate representation of a program) language into machine interface templates, which can then be translated into executable modules by the machine.

The PRM supports a symbolic interface to the create program instruction template. The PRM also supports the CPF symbol table and the break offset mapping table components of the program template.

The PRM does not support symbolic interfaces to the OIR (object information repository) and associated space. The caller of the PRM is responsible for formatting the file reference function and user-text information for the OIR and any information that is to go into the associated space.

Input To The PRM

Input to the PRM is a pointer to a control block containing data that controls the execution of the PRM. That control block contains pointers that point to other control blocks and data areas. They are:

- A pointer to the IRP text string to be processed
- A pointer to the UFCB (user file control block), which describes the listing file
- Pointers to areas that describe data targeted for the OIR
- A pointer that will contain addressability to the program being created
- Options that control the PRM and Create Program instruction

Output From The PRM

The output from the PRM is dependent upon the options specified. Output can consist of any or all of the following:

- A program module suitable for execution with addressability to the program module returned in a control block
- An instruction stream listing and an ODT (object definition table) summary with diagnostics
- A cross reference listing
- A dump of the machine interface program template produced by the PRM

GENERAL OVERVIEW

Program Resolution Monitor Modules

The PRM component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QPRROOTP–PRM Root: This module is the interface routine to the PRM. It creates and destroys work areas and handles exceptions that are signaled from other PRM modules.

> QPRPH01P–Phase 1 of the PRM: This module performs all lexical and syntactical analysis on the IRP source text string and it begins semantic analysis on the IRP source text string. QPRPH01P builds internal tables to be used by QPRPH02P in building the ODT, symbol table, and break offset mapping table portions of the program template. It also builds the instruction stream portion of the program template. This version of phase 1 supports 8191 ODT entries and builds the version 0 program template.

QPRPH11P–Alternate Phase 1 of the PRM: This module performs all lexical and syntactical analysis on the IRP source text string and it begins semantic analysis on the IRP source text string. QPRPH11P builds internal tables to be used by QPRPH02P in building the ODT, symbol table, and break offset mapping portions of the program template. It also builds the instruction stream portion of the program template. This version of phase 1 supports 32 767 ODT entries and builds the version 1 program template.

QPRPH02P–Phase 2 of the PRM: This module builds the break offset mapping table of the program template and the symbol table. QPRPH02P also uses modules QPRISTCK, QPRODTBL, QPRISTSM, QPRMICK, QPRMICK1, and QPRXRF.

QPRISTCK–Check Internal Symbol Table Semantics: This module performs semantic checks on the IST built by the QPRPH01P module. It checks for inconsistent entries and relational errors between entries. All program objects have definitions in the IST.

QPRODTBL–Build ODT from IST: This module builds the ODT portion of the program template from the internal symbol table.

QPRISTSM–Produce Object Summary: This module produces a summary listing of all program objects from the internal symbol table. It also lists any program object errors that were found in QPRPH01P or QPRISTCK.

QPRMICK–Check Machine Interface Instruction Operand Semantics for Version 0 Program Template: This module performs semantic checks on the instruction stream portion of the program template before the Create Program instruction is issued. QPRMICK takes each instruction in turn and checks the operands of that instruction for the required attributes.

QPRMICK1–Check Machine Interface Instruction Operand Semantics for Version 1 Program Template: This module performs semantic checks on the instruction stream portion of the program template before the Create Program instruction is issued. QPRMICK1 takes each instruction in turn and checks the operands of that instruction for the required attributes.

QPRXRF–PRM Cross Reference Listing: This module produces the cross reference listing for the named program objects in the IRP source.

QPRPH03P–Phase 3 of the PRM: This module completes the program template header, issues the Create Program instruction, adds any OIR data, and inserts the created program's addressability into a library.

The following two service modules are used by QPRPH01P, QPRISTCK, QPRODTBL, QPRISTSM, QPRMICK, QPRXREF, and QPRPH03P:

QPRCRASH–PRM Fatal Termination: This module is called whenever the PRM finds an internal error. The parameter passed to this module describes the error condition; QPRCRASH signals an exception that indicates a PRM failure.

QPRLIST–Output Routine for the PRM: This module is the PRM interface to the Common Data Management component. Depending on the input to QPRLIST, either it will print a line or it will eject a page and print two headings.

Figure PR-1 shows the components and functions that use the PRM and the components used by the PRM to perform its tasks.

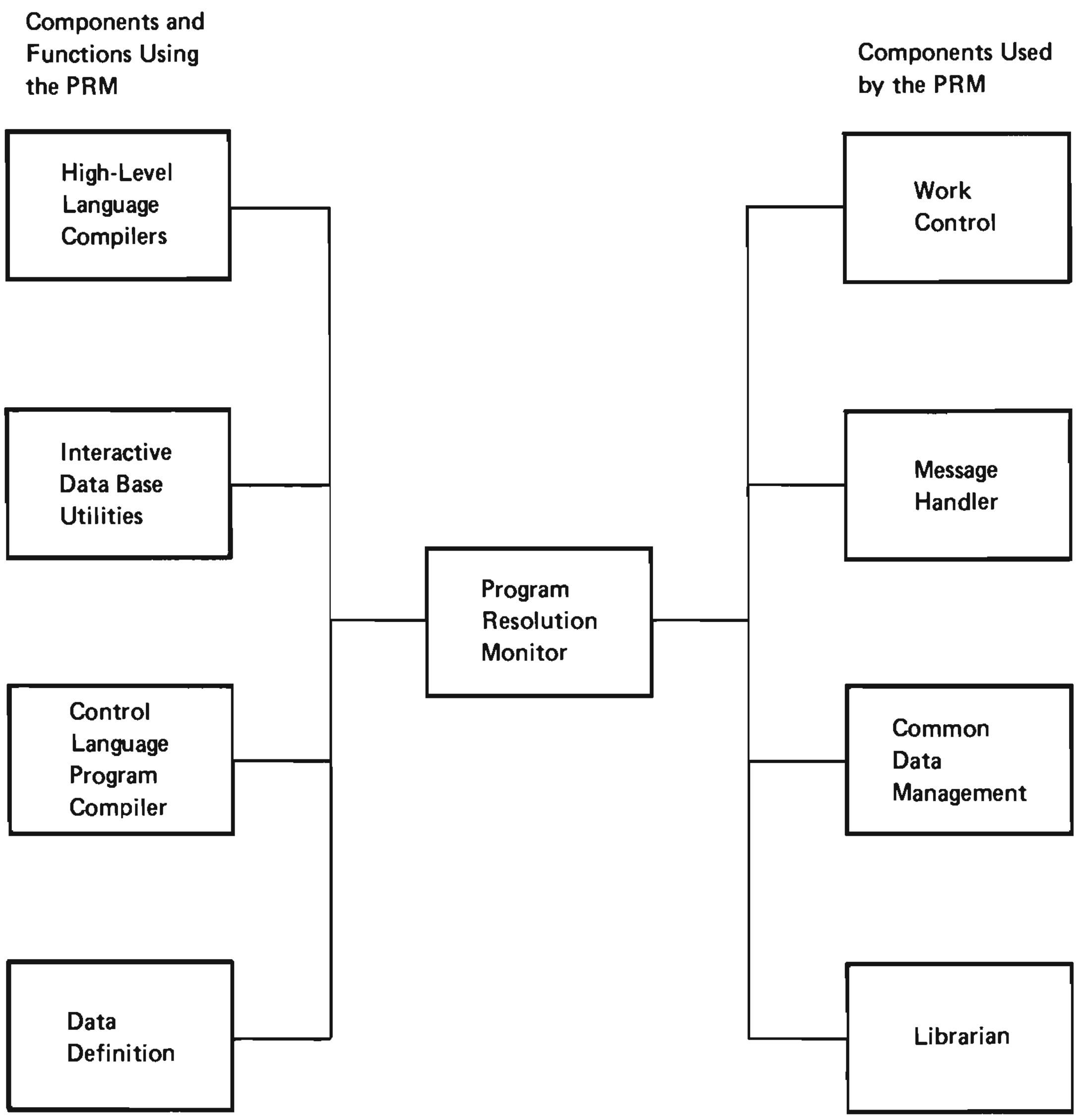

Components and Functions Using the PRM
Components Used by the PRM
High-Level Language Compilers
Interactive Data Base Utilities
Control Language Program Compiler
Data Definition
Program Resolution Monitor
Work Control
Message Handler
Common Data Management
Librarian

Figure PR-1. PRM and CPF Component Relationship Overview

PRM as Used by the RPG Compiler

Figure PR-2 and the following text describe an example of how the PRM would be used by the RPG compiler.

1 The RPG compiler calls QPRROOTP, passing a pointer to a control block. The control block contains pointers to other control blocks and data areas (IRP text string, options for the PRM). QPRROOTP performs the initialization procedures and calls QPRPH01P.

2 QPRPH01P parses and analyzes the IRP text string and builds tables to be used by QPRPH02P to build the program template. QPRPH01P produces an IRP text code listing. Control is returned to QPRROOTP, which then calls QPRPH02P.

If QPRPH01P determines that there are more than 8191 ODT entries, control is returned to QPRROOTP with an error indication. QPRROOTP then calls QPRPH11P to build the larger version 1 program template which supports 32 767 entries.

3 QPRPH02P, depending on the options specified and using other modules in the PRM, performs the following:

- Relational semantics checking of the symbol table
- Builds the ODV (object directory vector) and OES (object entry string) portions of the ODT using information from the internal symbol table and related areas
- Produces an object summary table listing for the objects in the symbol table
- Produces an IRP source cross reference listing
- Performs semantic checking on the operands, extender fields, and branch or indicator targets of the machine interface instructions
- Builds program template symbol table entries for each symbol table entry not noted by the compiler as having a temporary name
- Builds a program template break offset mapping entry for each entry in the breakpoint table

Control is returned to QPRROOTP, which then calls QPRPH03P.

4 QPRPH03P, if requested, resolves a system pointer to the library. It also completes the program template header, issues the Create Program instruction to build the encapsulated program, adds any OIR data, and inserts the program's addressability into the library. Control is returned to QPRROOTP, which in turn returns control to its caller.

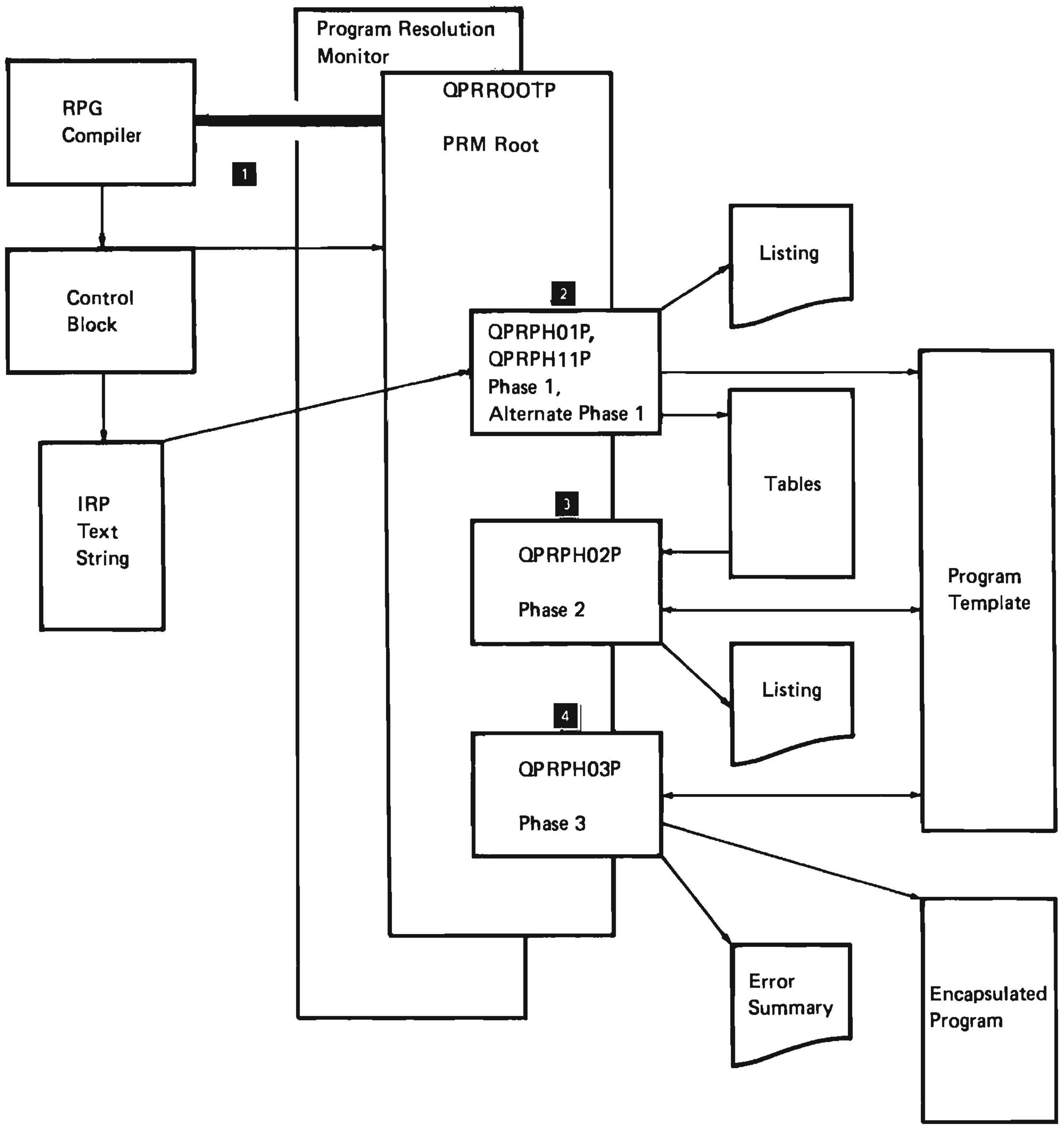


Figure PR-2. PRM as Used by the RPG Compiler

PRM Source Input and Its Associated Program Template

Figure PR-3 shows a formatted listing of the source input (IRP) to the PRM and the template produced by the PRM from that source. Figure PR-4 shows a dump of the machine interface program template produced by the PRM from the IRP shown in Figure PR-3.

```
5714SS1 R01M01                          GENERATED OUTPUT
SEQ    INST OFFSET  GENERATED CODE       . ... ...1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6

                                                    /* DECLARATIONS
00001                                               DCL DTAPTR PD INIT('B21')
00002                                               DCL SYSPTR PS INIT('$PRYR001')
00003                                               DCL SPCPTR PP INIT(C)
00004                                               DCL INSPTR PI INIT(B)
00005                                               DCL PTR PTR
00006                                               DCL SPCPTR APP(10)
00007                                               DCL DD I BIN(4)
00008                                               DCL DD C01 CHAR(1)
00009                                               DCL DD C02 CHAR(2)
00010                                               DCL DD C03 CHAR(3)
00011                                               DCL DD C04 CHAR(4)
00012                                               DCL DD C05 CHAR(5)
00013                                               DCL DD C07 CHAR(7)
00014                                               DCL DD C08 CHAR(8)
00015                                               DCL DD C16 CHAR(16)
00016                                               DCL DD C32 CHAR(32)
00017                                               DCL DD C34 CHAR(34)
00018                                               DCL DD C48 CHAR(48)
00019                                               DCL DD C64 CHAR(64)
00020                                               DCL DD C CHAR(128)
00021                                               DCL DD Z ZND(5,0)
00022                                               DCL DD P PKD(5,0)
00023                                               DCL DD AN(10) BIN(4)
00024                                               DCL DD AC(10) CHAR(5)
00025                 Program                       DCL DD AI(10) BIN(4)
00026                                               DCL DD BI BIN(2) BAS(PP)
00027                 Template                      DCL DD BC CHAR(128) BAS(*)
                                                    /* SAMPLE INSTRUCTIONS
00028  0001 000004  3043 0007 0007 0007  B:         ADDN I,I,I
00029  0002 00000C  1C23 1472 0014 0014             ADDLC(B) C,C,C / ZC(B),ZNTC(B),NTZC(B),NTZNTC(B)
                    0014 001C 001C 001C
                    001C
00030  0003 00001E  1A43 1000 0016 2004             ADDN(RI) P,4,0439 / ZC(C01)
                    21B7 0008
00031  0004 00002A  1193 801B 0003 0014             AND(S) PP-°BC,C
00032  0005 000032  1011 001C                       B B
00033  0006 000036  1CC2 1240 0014 0007             CMPBLA(B) C,I / HI(B),LO(B),EQ(B)
                    001C 001C 001C
00034  0007 000044  1C46 1240 0007 0015             CMPNV(B) I,Z / HI(B),LO(B),EQ(B)
                    001C 001C 001C
00035  0008 000052  10A3 0014 2009 001D             CVTNC C,9,X'02001F00000000'
00036  0009 00005A  10B6 0007 2005                  CPYBRA I,5
00037  000A 000060  1096 0007 001E                  CPYHEXNZ I,'4'
00038  000B 000066  1C42 1240 0015 0016             CPYNV(B) Z,P / POS(B),NEG(B),ZER(B)
                    001C 001C 001C
00039  000C 000074  184F 1000 0015 200A             DIV(I) Z,10,4 / POS(C01)
                    2004 0008
00040  000D 000080  10E3 0014 001F 0020             EDIT C,Z'99.99','B1B2'
00041  000E 000088  10CE 0015 000C                  EXCHBY Z,C05
00042  000F 00008E  1C9B 4000 0014 0014             XOR(B) C,C,C / ZER(B)
                    0014 001C
00043  0010 00009A  18D2 4000 0004 0004             CMPPTRA(I) PI,PI / EQ(C01)
                    0008
00044  0011 0000A4  0112 0002 0003                  CRTCTX PS,PP
00045  0012 0000AA  0162 0002 0021                  RENAME PS,'CHAR'
00046  0013 0000B0  0164 0002 0011 0000             RSLVSP PS,C34,*,C02
                    0009
00047  0014 0000BA  0260                            PEND
```

Figure PR-3 (Part 1 of 3). Output and IRP Listing of the PRM

Symbolic Explanation of ODV/OES

```
5714SS1 R01M01
 ODT  ODT NAME                                 ATTRIBUTES AND ODV/OES ENTRIES

 0001 PD                                         POINTER OBJECT,STATIC,INITIAL VALUE,DATA POINTER.
                                                 18030004/0400010003C2F2F1
 0002 PS                                         POINTER OBJECT,STATIC,INITIAL VALUE,SYSTEM POINTER.
                                          ODV    1802000C/0400010201000000085BD7D9E8D9F0F0F1
 0003 PP                                         POINTER OBJECT,STATIC,INITIAL VALUE,SPACE POINTER.
                                                 1801001D/040014
 0004 PI                                         POINTER OBJECT,STATIC,INITIAL VALUE,INSTRUCTION POINTER.
                                                 18040020/04001C ——— OES portion of program template
 0005 PTR                                        POINTER OBJECT,STATIC.
                                                 10000000/
 0006 APP                                        POINTER OBJECT,STATIC,ARRAY(10),SPACE POINTER.
                                                 18010023/200000000A0000
 0007 I                                          DATA OBJECT,STATIC,BINARY(4),INTERNAL.
                                                 00000004/
 0008 C01                                        DATA OBJECT,STATIC,CHARACTER(1),INTERNAL.
                                                 00040001/
 0009 C02                                        DATA OBJECT,STATIC,CHARACTER(2),INTERNAL.
                                                 00040002/
 000A C03                                        DATA OBJECT,STATIC,CHARACTER(3),INTERNAL.
                                                 00040003/
 000B C04                                        DATA OBJECT,STATIC,CHARACTER(4),INTERNAL.
                                                 00040004/
 000C C05                                        DATA OBJECT,STATIC,CHARACTER(5),INTERNAL.
                                                 00040005/
 000D C07                                        DATA OBJECT,STATIC,CHARACTER(7),INTERNAL.
                                                 00040007/
 000E C08                                        DATA OBJECT,STATIC,CHARACTER(8),INTERNAL.
                                                 00040008/
 000F C16                                        DATA OBJECT,STATIC,CHARACTER(16),INTERNAL.
                                                 00040010/
 0010 C32                                        DATA OBJECT,STATIC,CHARACTER(32),INTERNAL.
                                                 00040020/
 0011 C34                                        DATA OBJECT,STATIC,CHARACTER(34),INTERNAL.
                                                 00040022/
 0012 C48                                        DATA OBJECT,STATIC,CHARACTER(48),INTERNAL.
                                                 00040030/
 0013 C64                                        DATA OBJECT,STATIC,CHARACTER(64),INTERNAL.
                                                 00040040/
 0014 C                                          DATA OBJECT,STATIC,CHARACTER(128),INTERNAL.
                                                 00040080/
 0015 Z                                          DATA OBJECT,STATIC,ZONED(5,0),INTERNAL.
                                                 00020005/
 0016 P                                          DATA OBJECT,STATIC,PACKED(5,0),INTERNAL.
                                                 00030005/
 0017 AN                                         DATA OBJECT,STATIC,BINARY(4),INTERNAL,ARRAY(10).
                                                 0800002A/6000040000000A0000
 0018 AC                                         DATA OBJECT,STATIC,CHARACTER(5),INTERNAL,ARRAY(10).
                                                 08040033/6000050000000A0000
 0019 AI                                         DATA OBJECT,STATIC,BINARY(4),INTERNAL,ARRAY(10).
                                                 0800003C/6000040000000A0000
 001A BI                                         DATA OBJECT,BASED(PP),BINARY(2),INTERNAL.
                                                 0A000045/5000020003
```

Figure PR-3 (Part 2 of 3). Output and IRP Listing of the PRM

```
5714SS1 R01M01
ODT   ODT NAME

0018 AC            24*
0019 AI            25*
0017 AN            23*
0006 APP           6*
001C B             4 28* 29 29 29 29 32 33 33 33 34 34 34 38 38 38 42
001B BC            27* 31
001A BI            26*
0014 C             3 20* 29 29 29 31 33 35 40 42 42 42
0008 C01           8* 30 39 43
0009 C02           9* 46
000A C03           10*
000B C04           11*
000C C05           12* 41
000D C07           13*
000E C08           14*
000F C16           15*
0010 C32           16*
0011 C34           17* 46
0012 C48           18*
0013 C64           19*
0007 I             7* 28 28 28 33 34 36 37
0016 P             22* 30 38
0001 PD            1*
0004 PI            4* 43 43
0003 PP            3* 26 31 44
0002 PS            2* 44 45 46
0005 PTR           5*
0015 Z             21* 34 38 39 41
```

Cross Reference

Figure PR-3 (Part 3 of 3). Output and IRP Listing of the PRM

```
5714SS1 R01M01                              GENERATED OUTPUT
 OFFSET         MI TEMPLATE DISPLAY

 00000000       00000CBD  00000000  0201D7D9  D4E3C5E2  E3404040  40404040  40404040  40404040
 00000020       40404040  40404040  40000000  00000000  00000039  00000000  00000000  00000000
 00000040       00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
 00000060       000000FC  00000000  00000000  00140021  00000100  000001BC  00000244  00000000
 00000080       00000000  00000000  00000000  000009F1  000002CC  00000000  00000000  00000000
 000000A0       00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
 000000C0       00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
 000000E0       00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
 00000100       000000BC  30430007  00070007  1C231472  00140014  0014001C  001C001C  001C1A43
 00000120       10000016  200421B7  00081193  801B0003  00141011  001C1CC2  12400014  0007001C
 00000140       001C001C  1C461240  00070015  001C001C  001C10A3  00142009  001D10B6  00072005
 00000160       10960007  001E1C42  12400015  0016001C  001C001C  184F1000  0015200A  20040008
 00000180       10E30014  001F0020  10CE0015  000C1C9B  40000014  00140014  001C18D2  40000004
 000001A0       00040008  01120002  00030162  00020021  01640002  00110000  00090260  00000088
 000001C0       18030004  1802000C  1801001D  18040020  10000000  18010023  00000004  00040001
 000001E0       00040002  00040003  00040004  00040005  00040007  00040008  00040010  00040020
 00000200       00040022  00040030  00040040  00040080  00020005  00030005  0800002A  08040033
 00000220       0800003C  0A000045  02040080  30000001  6804004A  68040054  68020058  6804007A
 00000240       68040081  00000088  04000100  03C2F2F1  04000102  01000000  085BD7D9  E8D9F0F0
 00000260       F1040014  04001C20  0000000A  00006000  04000000  0A000060  00050000  000A0000
 00000280       60000400  00000A00  00500002  00034200  0702001F  00000000  420001F4  42021FF0
 000002A0       F0F0F0F0  F0F0F0F0  F0F0F0F0  F0F0F0F0  F0F0F0F0  F0F0F0F0  F0F0F9F9  F9F94200
 000002C0       04C2F1C2  F2420004  C3C8C1D9  00000233  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 000002E0       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000300       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000320       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000340       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000360       00000922  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000380       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 000003A0       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 000003C0       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 000003E0       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000400       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000420       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000440       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000460       000009E8  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  0000097A  FFFFFFFF
 00000480       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 000004A0       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 000004C0       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  000009DE  FFFFFFFF  FFFFFFFF
 000004E0       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
 00000500       FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
```

Figure PR-4 (Part 1 of 2). Machine Interface Program Template

```
00000520      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000540      FFFFFFFF  0000096F  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  00000964  FFFFFFFF
00000560      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000580      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  000009AD  00000903  FFFFFFFF
000005A0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000005C0      FFFFFFFF  FFFFFFFF  0000094E  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000005E0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000600      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000620      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000640      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  00000990
00000660      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  00000985  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000680      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  00000943  FFFFFFFF  000009A4
000006A0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000006C0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000006E0      FFFFFFFF  FFFFFFFF  000008E4  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000700      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000720      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000740      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000760      0000092D  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000780      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000007A0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  000009C0  FFFFFFFF  FFFFFFFF
000007C0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000007E0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000800      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000820      FFFFFFFF  FFFFFFFF  FFFFFFFF  00000917  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000840      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000860      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000880      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  000009D4
000008A0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000008C0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000008E0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000900      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000920      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000940      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000960      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000980      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  000008D0  FFFFFFFF
000009A0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000009C0      FFFFFFFF  FFFFFFFF  00000959  FFFFFFFF  000008EE  FFFFFFFF  FFFFFFFF  FFFFFFFF
000009E0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000A00      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000A20      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000A40      000009B6  FFFFFFFF  FFFFFFFF  0000099B  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000A60      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000A80      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000AA0      FFFFFFFF  0000090E  FFFFFFFF  FFFFFFFF  000008F8  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000AC0      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000AE0      FFFFFFFF  FFFFFFFF  000008DA  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000B00      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000B20      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000B40      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000B60      00000938  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  000009CA
00000B80      FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00000BA0      0001C002  D7C4FFFF  FFFF0002  C002D7E2  FFFFFFFF  0003C002  D7D7FFFF  FFFF0004
00000BC0      C002D7C9  FFFFFFFF  0005C003  D7E3D9FF  FFFFFF00  06C003C1  D7D7FFFF  FFFF0007
00000BE0      C001C9FF  FFFFFF00  08C003C3  F0F1FFFF  FFFF0009  C003C3F0  F2FFFFFF  FF000AC0
00000C00      03C3F0F3  FFFFFFFF  000BC003  C3F0F4FF  FFFFFF00  0CC003C3  F0F5FFFF  FFFF000D
00000C20      C003C3F0  F7FFFFFF  FF000EC0  03C3F0F8  FFFFFFFF  000FC003  C3F1F6FF  FFFFFF00
00000C40      10C003C3  F3F2FFFF  FFFF0011  C003C3F3  F4FFFFFF  FF0012C0  03C3F4F8  FFFFFFFF
00000C60      0013C003  C3F6F4FF  FFFFFF00  14C001C3  FFFFFFFF  0015C001  E9FFFFFF  FF0016C0
00000C80      01D7FFFF  FFFF0017  C002C1D5  FFFFFFFF  0018C002  C1C3FFFF  FFFF0019  C002C1C9
00000CA0      FFFFFFFF  001AC002  C2C9FFFF  FFFF001B  C002C2C3  FFFFFFFF  00014001  C2000000
```

Figure PR-4 (Part 2 of 2). Machine Interface Program Template

INTRODUCTION

The prompter component of the CPF (control program facility) presents prompt displays to be used in building valid CL commands. To do this, the prompter uses the information that is in the CDO (command definition object) of the command being entered. Pressing the CF4 Key, entering a question mark before the command name, or entering selective prompt characters before a parameter invokes the prompter.

The prompter interfaces with the command analyzer to build and validate a command that conforms to the user-defined options. The command is usually entered in a string format. The prompter and command analyzer, however, build the command internally in a variable length positional list and validate the user-defined parameter values in that format. The prompter can then convert the positional list back to a string format for insertion, system logging, and displaying.

The prompter provides the following services to the user:

- A display of each command and its parameters from the CDO with display space provided so that the user can enter the desired parameters
- A display of any parameter values entered, either initially with the command or during prompting and value entry
- An automatic or user-defined entry of default parameter values from the CDO
- Separate display list to enter values for parameters that can accept a variable length series of values, and the insertion of a user-defined list of values into the command parameter display and command analyzer positional list
- Separate display lists of allowable values from the CDO for parameters that can have one or more fixed values, and provides for entering and the insertion of those values into the command parameter display and command analyzer positional list
- A display showing a list that the prompter recognizes of the console function keys and their assigned values
- A display showing a list of each command analyzer message pending on the command currently being prompted for
- A response to pressing a function key. The function keys provide the following services:

CF1	Terminate the prompting cycle
CF2	Back up to previous display
CF4	A parameter display for a command embedded within the command being entered
CF13	A display of function key assignments
CF14	A display of the command in string format
CF15	An error message display
CF16	Enter the completed command
CF18	A redisplay of the initial parameter display with all default parameter values shown. For selective prompting, a redisplay of the initial parameter display with all the initial rules
HELP	A display of function key assignments

GENERAL OVERVIEW

Prompter Modules

The prompter component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QPTPARML–Prompter Control: This module controls the prompting cycle by calling QPTSETUP for initialization, QPTPRCSS for the user interface cycle, QPTCHECK for validity checking, and then transfers control to the command analyzer for command execution if in execute mode, otherwise returns control to the caller.

QPTSETUP–Main Processor Setup Routine: This module initializes a prompting list space that contains addressability to command definition, positional list, and user-entered data required for the prompting cycle.

QPTCHECK–Command Check Routine: This module is called to housekeep the command positional list, delete all error messages in the queue, and to call QPTSTRNG to generate the string form of the command. It also interfaces with the command analyzer to validity check the command.

QPTPRCSS–Parameter and List Screen Processor: This module generates the parameter and list screens for the user to enter the desired values. It calls QPTPERMV to generate the permissible values display. QPTPRCSS calls QPTGTINP to get the user-defined values and QPTVLINP to housekeep the positional list and to call the command analyzer to syntax check the command. QPTPRCSS calls QPTERMSG to process any error messages.

QPTPERMV–Permissible Value Display Processor: This module generates the permissible value display. It calls QPTGTINP, QPTVLINP, and QPTERMSG for the same purposes that QPTPRCSS called them.

QPTGTINP–Get Input from Device and Process: This module acquires user input from the display and updates the positional list.

QPTDFT–Build Default Entry in Positional List: This module builds a default entry in the positional list and supplies a default value for the entry.

QPTVLINP–Validate Display Input Via Command Analyzer: This clears the positional list flags, deletes obsolete error messages, and interfaces with the command analyzer to validity check the command in its current form.

QPTERMSG–Error Message Processor: This module retrieves error messages from the queue that are pertinent to the current command.

QPTKYPRC–Function Key Processor: This module responds to any user-function key by providing the requested function, or by calling QPTPFKRV to generate the function key review display, QPTERREV to generate the error message review display, or QPTCMSRV to generate the command string review display.

QPTPFKRV–Function Key Review Screen Processor: This module generates the function key review display to remind the user of the key functions available. QPTPFKRV also reminds the user of special prompter operators.

QPTERREV–Error Message Review Display Processor: This module calls QPTERMSG to retrieve error messages pertinent to the current command status and displays these messages to the user.

QPTCMSRV–Command String Review Display Processor: This module calls QPTSTRNG to build the string form of the command and displays this string form to the user.

QPTSTRNG–Command String Creation: This module builds a string form of the command based on the current command positional list.

Prompter Invocation Paths

Figure PT-1 shows the three possible paths to invoke the prompter component; it also shows the CPF components that support access to the console device and to message handling.

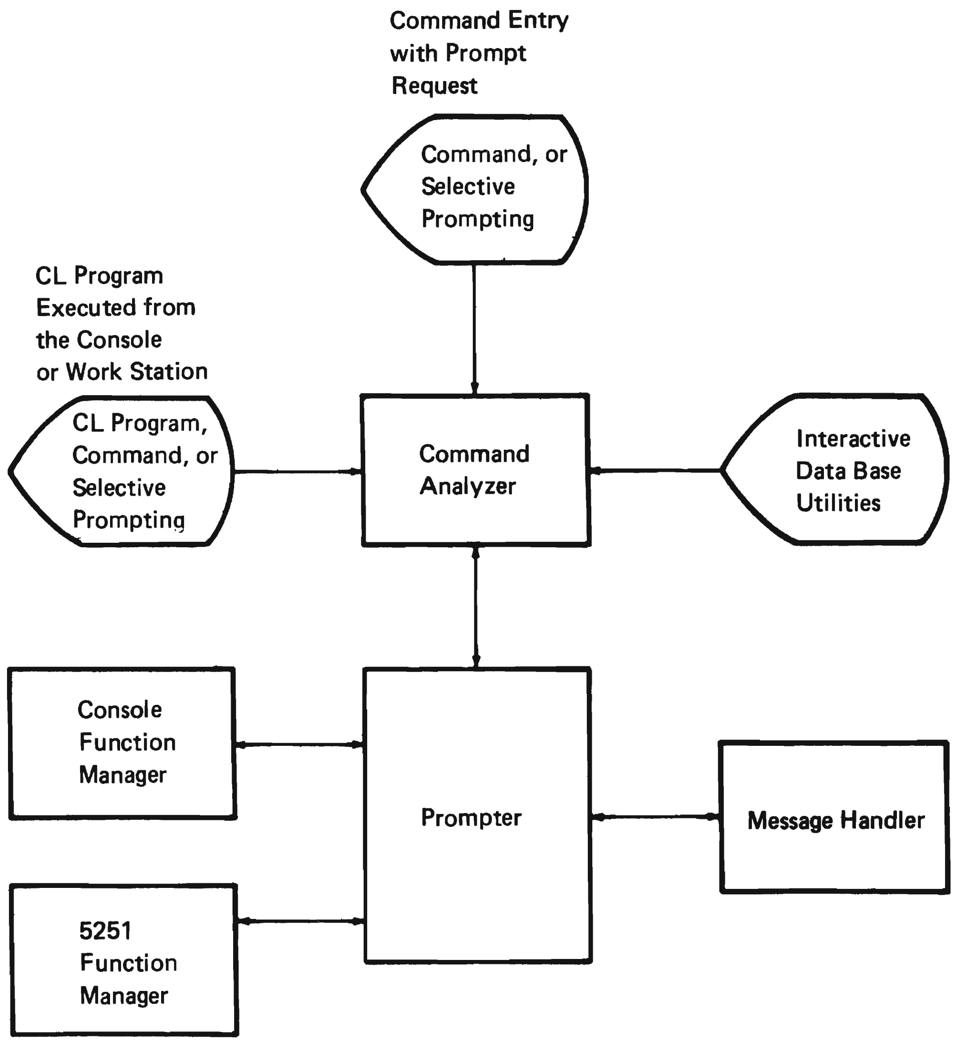

Command Entry with Prompt Request
Command, or Selective Prompting
CL Program Executed from the Console or Work Station
CL Program, Command, or Selective Prompting
Command Analyzer
Interactive Data Base Utilities
Console Function Manager
Prompter
Message Handler
5251 Function Manager

Figure PT-1. Prompter Overview

Prompter Invocation and Control Overview

Figure PT-2 and the following text describe the module relationship in the invocation and control of the prompter component.

1 No matter how the prompter is invoked, QPTPARML is called by command analyzer to control the prompter processing. QPTPARML calls QPTSETUP to perform initialization procedures.

2 When QPTSETUP gets control, it creates and initializes a prompting list space. The prompting list provides access to the command definition and positional list information needed to drive all prompter displays. It also contains a history of user-defined data and error messages associated with the parameter values entered.

QPTSETUP inserts any initial command analyzer messages in the prompting list and establishes pointers to the positional list passed by the command analyzer and to the CDO for the command being built. QPTSETUP also saves the initial input values for selective prompting.

3 After the prompting list has been initialized, QPTPARML begins the prompting cycle by calling QPTPRCSS. QPTPRCSS generates the primary command prompt displays (see Figure PT-3).

4 When the prompting cycle is finished, QPTPARML calls QPTCHECK to reset certain flags in the command analyzer positional list, and deletes all error messages in the queue for the command just built. QPTCHECK then calls QPTSTRNG to rebuild the command in string format from the current command analyzer positional list. QPTCHECK calls the command anlayzer to perform a final validity check on the command.

5 Control is returned to QPTPARML to destroy the prompting space and close the console device file. If specified by the user, control is transferred to the command analyzer to execute the command.

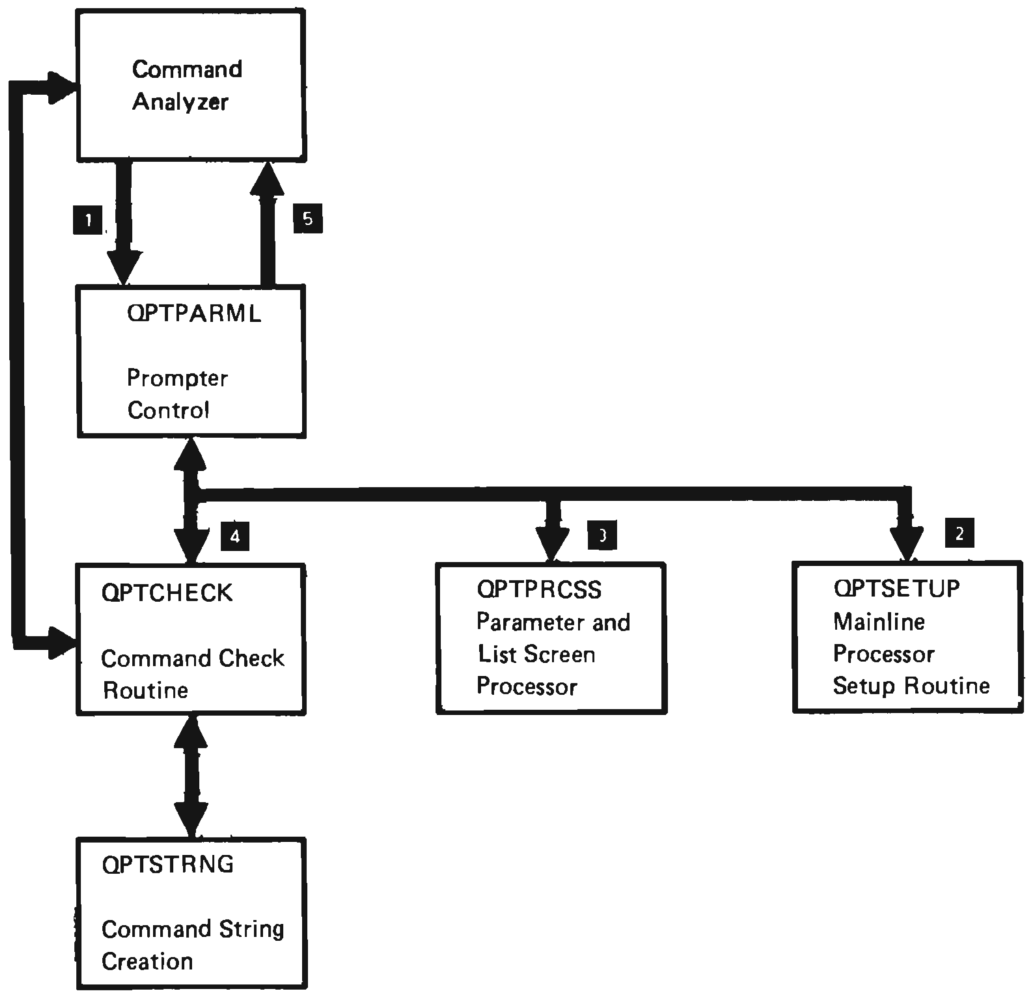


Figure PT-2. Prompter Invocation and Control Overview

Initial Parameter Display, List of Values Display, and Permissible Values Display Overview

Figure PT-3 and the following text show the module relationship to control the primary prompt display generation and to read and validate user-defined parameter values and value lists.

1 QPTPARML calls QPTPRCSS to generate the parameter display based on the command definition of the command being entered. If requested, the list of values display is generated by QPTPRCSS so that a list of values can be entered for the parameter.

2 The permissible values display for a keyword can be requested by using the special display symbol input. QPTPRCSS calls QPTPERMV to generate this display.

3 Both QPTPRCSS and QPTPERMV process user-defined values by first calling QPTGTINP to get the user-defined values from the device file and to update the parameter positional list.

4 QPTPRCSS and QPTPERMV then call QPTVLINP to delete the current positional list flags, to delete any messages on the queue for the parameter values being changed, and to call command analyzer to validate the updated command positional list. QPTVLINP will supply the default value or the initial value for parameters that are blanked out. QPTDFT is invoked to supply the default values for all parameters that are not selectively prompted.

5 Both QPTPRCSS and QPTPERMV call QPTERMSG to process any error messages resulting from the command build cycle.

The preceding processing cycle continues until one of the following is signaled by pressing a function key (see Figure PT-4):

- The user is satisfied with the command as entered.
- The user wants to abort the processing cycle.
- The user wants to invoke a prompter support function.
- The user wants to reset all command parameters to their default values or their original values.

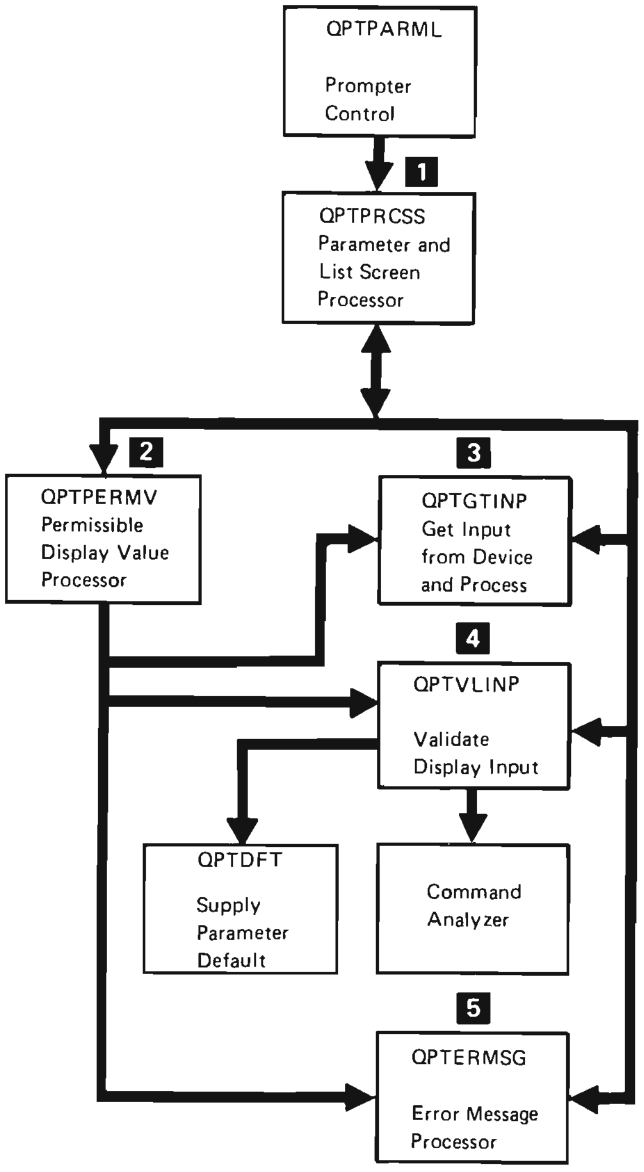


Figure PT-3. Initial Parameter Display, List of Values Display, and Permissible Values Display Overview

Function Key Processing

QPTKYPRC is called whenever a user interrupts the prompt display cycle by pressing a function key. It intercepts the key and either performs the requested function or indicates to the calling module what function is to be performed.

Figure PT-4 and the following text show the module relationship when a function key is pressed.

1 If the Enter key is pressed and the last parameter screen is displayed, QPTKYPRC performs the same function as when CF16 is pressed. If the last parameter screen is not displayed, QPTKYPRC determines the next screen to be displayed.

2 If CF1 is pressed, QPTKYPRC calls QPTSTRNG to rebuild the command string, and then signals exception CPF6801 to the caller of the prompter. If the command being prompted for is to be logged, a type command message must be sent to the program's message queue that invoked the command analyzer with command prompting requested. The type command message contains the rebuilt command string.

3 If CF2 is pressed, QPTKYPRC indicates that QPTPRCSS is to display the previous screen.

4 If CF4 is pressed, QPTKYPRC calls the command analyzer. The command analyzer re-invokes the prompter to prompt for the embedded command.

5 If CF13 or the Help key is pressed, QPTKYPRC calls QPTPFKRV to generate a display of the function key options available to the user.

6 If CF14 is pressed, QPTKYPRC calls QPTCMSRV to generate the command string review display. QPTCMSRV then calls QPTSTRNG to build the string format from the current positional list, and displays the command being entered in string format.

7 If CF15 is pressed, QPTKYPRC calls QPTERREV to generate an error message display.

8 If CF16 is pressed, QPTKYPRC checks all previously displayed screens for errors. If an error is found, QPTKYPRC indicates that QPTPRCSS is to display the first screen that has an error. If no errors are found, QPTKYPRC indicates that prompting is complete, and that QPTPRCSS is to return to QPTPARML for a final check of the command by QPTCKECK.

9 If CF18 is pressed, QPTKYPRC calls QPTDFT to supply default values for all parameters that are not selectively prompted. The defaults for selectively prompted parameters are obtained from user-defined values stored in the positional list by QPTKYPRC.

10 If the Roll Up key is pressed and a list screen is displayed, QPTKYPRC indicates that the next list screen is to be displayed. If not on a list screen, QPTKYPRC indicates that the same screen is to be displayed.

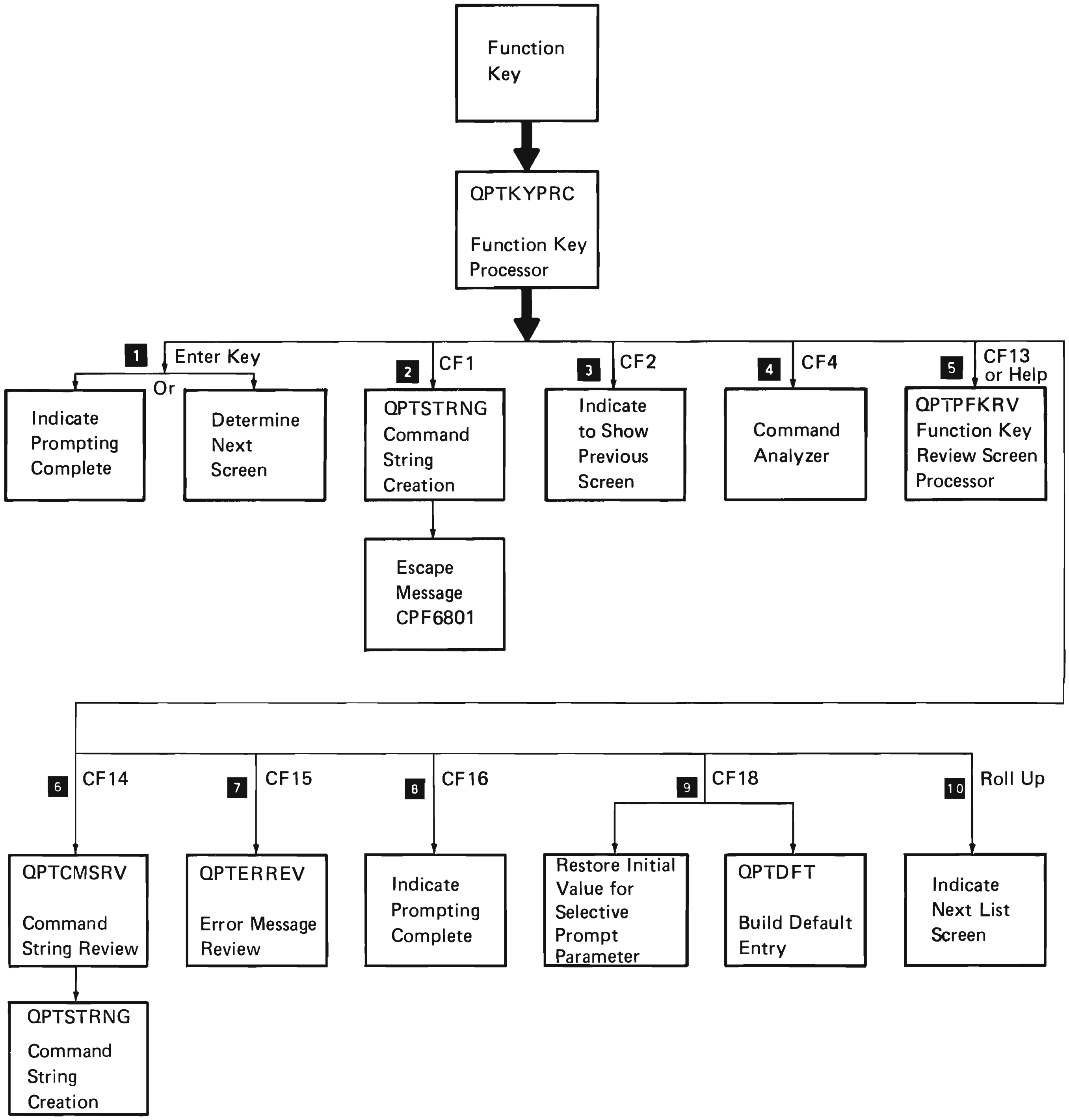

Function Key
QPTKYPRC Function Key Processor
1 Enter Key
Or
Indicate Prompting Complete
Determine Next Screen
2 CF1
QPTSTRNG Command String Creation
Escape Message CPF6801
3 CF2
Indicate to Show Previous Screen
4 CF4
Command Analyzer
5 CF13 or Help
QPTPFKRV Function Key Review Screen Processor
6 CF14
QPTCMSRV Command String Review
QPTSTRNG Command String Creation
7 CF15
QPTERREV Error Message Review
8 CF16
Indicate Prompting Complete
9 CF18
Restore Initial Value for Selective Prompt Parameter
QPTDFT Build Default Entry
10 Roll Up
Indicate Next List Screen

Figure PT-4. Function Key Processing

Error Message Display Overview

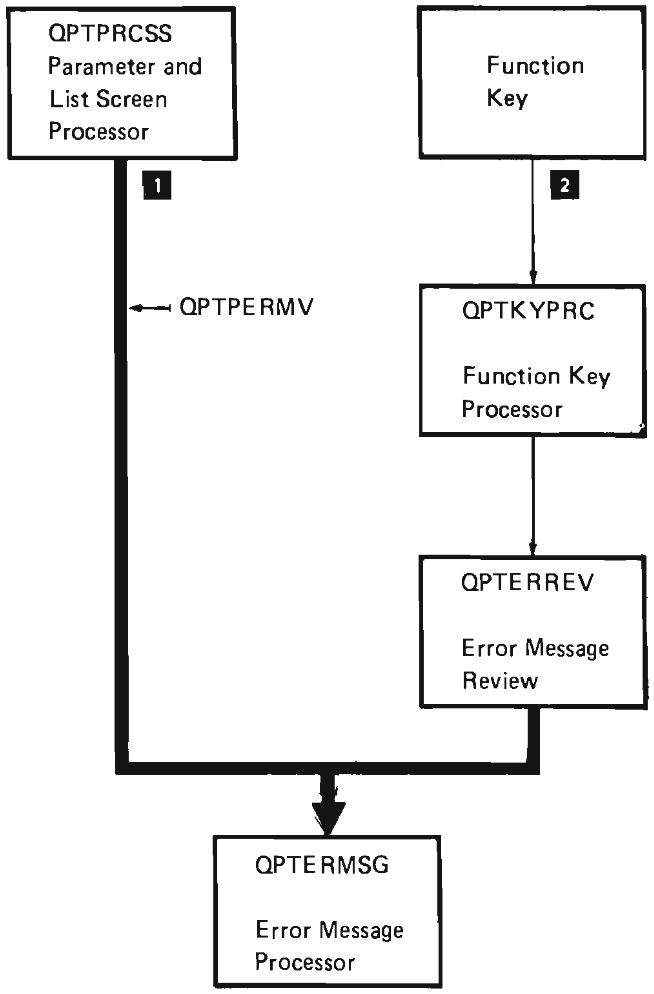
Figure PT-5 and the following text show the module relationship during the display of error messages. During the command build cycle, the prompter processes errors detected in the user-defined display and function key use, syntax errors detected by the command analyzer, and system errors signaled for the prompter. The prompter makes the error messages available to the user in two ways:

1 Each display includes an error message line at the bottom of the display; the modules and displays are:

QPTPRCSS	Initial parameter display
QPTPRCSS	List of values display
QPTPERMV	Permissible values display
QPTCMSRV	Command string review display
QPTPFKRV	Function key review display
QPTERREV	Error message review display (This uses the entire display, for the error message display, not just the bottom line.)

QPTPRCSS calls QPTERMSG to retrieve all the error messages associated with the current display and writes them to the error block for the current display. The top message in the error block is displayed at the bottom of the current display. The roll function can be used to view additional messages associated with the display. The error block is a message subfile.

2 If all messages associated with the current display are to be viewed, the CF15 key is pressed, which invokes QPTERREV through QPTKYPRC to generate the error message review display (see Figure PT-4). QPTERREV in turn calls QPTERMSG, which retrieves all messages from the queue.



QPTPRCSS Parameter and List Screen Processor
Function Key
1
2
QPTPERMV
QPTKYPRC Function Key Processor
QPTERREV Error Message Review
QPTERMSG Error Message Processor

Figure PT-5. Error Message Display Overview

Reclaim/Damage Notification

INTRODUCTION

The reclaim/damage notification component of the CPF (control program facility) is responsible for recovery of data at the object level. The component's primary purpose is to give the user information on both relational and physical object damage. To provide this facility, this component is divided into the following functions:

- The reclaim storage function gives the user an interface to clean up object relationships and to recover storage for permanent objects no longer addressable through the CPF command interface. In some cases, damaged objects will be deleted (subsystem descriptions, for example), while other damaged objects will be repaired to minimize potential data loss for the user (libraries or data base files, for example). Unnecessary or duplicate IBM-supplied objects will be eliminated to free physical auxiliary storage.
- Damage notification and logging provides the user with identification information and probable recovery procedures upon reference to physically damaged objects.

GENERAL OVERVIEW OF RECLAIM

Reclaim Modules

The reclaim portion of this component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QRCLAIM–Reclaim Driver: This module is responsible for controlling the overall reclaim utility. It creates the necessary space and index objects used by the other modules, forms the controlling routine for most status messages, and verifies the initial condition requirements for undamaged libraries and system quiesced status.

-->QLIRCLIB–Handle Damaged Libraries: This module checks the recovery library for library recovery objects and deletes the partially completed libraries and their library recovery object. It also rebuilds any library whose context associated space is damaged. All other objects that comprise a library will be checked for damage. If damage is detected, the object will be deleted and re-created, saving any usable information.

QRCALOWN–Guarantee All Objects Are Owned: This module verifies that all objects have a valid owner. To perform this, all profiles are checked for damage and are either deleted or the function is terminated, depending on the profile that is found to be damaged. After damaged profiles have been eliminated, the module issues the Reclaim Vertical Microcode instruction to return a list of all ownerless and any duplicate machine context objects. Duplicate machine context objects that are libraries and profiles are deleted. Logical unit descriptions, control unit descriptions, and line descriptions that are not addressable through the machine context are also deleted, and all ownerless objects are given an owner. For most objects the owner is determined from a table (reclaim definition table); for message queues,the owner is determined based on the type of message queue. Other objects are given to the security officer profile.

QRCSEPOB–Separate Objects Into Classes: This module separates the owned objects in a particular profile into the various classes for handling. Objects that are formats and directories are processed by storing information on them into a table which is processed after all other objects have been handled. Information on the secondary pieces of composite objects are stored into an index for handling when the primary piece of the object is processed. Those objects that have an invalid subtype are destroyed and the system operator is informed. Primary pieces of composite objects are handled by passing information on the object to the QRCOMPST module. All other objects are classified as simple objects and are processed by the QRCSIMPL module.

QRCSIMPL–Simple Object Handling: This module verifies that objects are properly addressable through the correct context. Duplicates are deleted or inserted into a special reclaim library (QRCL), depending on the type/subtype of the object, the owner of the object, and the necessity to retain the object on the system (for example, save restore authority objects can be deleted).

QRCOMPST–Composite Object Handling: This module verifies that primary pieces of composite objects are properly addressable and contain the necessary pieces to be valid objects. Improperly addressable primary pieces are moved into a library or are deleted if they were last in temporary libraries. The secondary pieces are checked and information on them is used to update the index of secondary objects. Damaged objects are either repaired (as in the case of libraries and certain files) or deleted (as in the case of subsystem descriptions). Ownership and authorities of all secondary pieces are updated to be consistent with their primary pieces.

QLIRCLIB–Handle Damaged Libraries: This module is called to repair damaged libraries.

QDBRCLMA–Verify Secondary Pieces of Files: This module returns a list of all secondary pieces found for a file. It updates the ownership and authorities of all secondary pieces to be consistent with the primary. Also, any member control blocks that are not in the correct library are moved and renamed to be consistent with the primary.

QRCLENUP–Clean Up Dangling Pieces and Process Format and Directories: This module processes the secondary pieces of composite objects that do not have a primary piece associated with them. With the exception of files, dangling secondary pieces are deleted. For files, those secondary pieces that contain meaningful information are rebuilt into valid file structures that the user can access to retrieve the data. After dangling secondary pieces are processed, the formats and directories are processed. Those that no longer contain meaningful data are deleted.

QDBRCLMB–Process Dangling Pieces of Files: This module creates valid file structures for those pieces that contain lost user data (data spaces or members that address indexes/data spaces).

QDBRCLMC–Process Formats and Directories: This module determines whether the formats and directories contain valid file information and deletes those that do not. Any remaining file recovery objects that could not be processed successfully by data base recovery are deleted from the system.

QRCINSRT–Insert Object Into the Reclaim Library: This module inserts object into the reclaim library. For those objects that are duplicates in this library, the name is changed to an alias, and the original name information is saved as object description text information.

QRCDEBUG–Debug Dump for Reclaim: This module outputs a dump of the actions taken for all objects that are destroyed, that change owners, or that change libraries. Additionally, the profile and libraries in the system are listed. This information is used only for problem determination on the reclaim function. As such, it is invoked only when the system is servicing the arbiter process (documentation requests that the user not use the facility as it only provides information useful to IBM).

Reclaim Storage Function

The reclaim storage function is used to recover physical storage that has become unaddressable through normal CPF interfaces. This may be due to damage to libraries and composite objects, or due to APAR (authorized program analysis report) conditions that cause incorrect addressability and conditions through which certain objects cannot be deleted from the system.

The recovery of physical storage results in two possible object conditions. One is the case in which an object either was residing in the wrong library, or has become lost from a library and has been reinserted into the proper library or inserted into a library designated by the reclaim function. The second case occurs when duplicate objects or certain types of damage are detected; internal duplicate objects are deleted from the system.

Object Addressability

Objects on the System/38 are addressable by using one of the following schemes:

- Context interface: This scheme provides addressability through libraries. The user (via resolves) states what library or list of libraries are to be searched to find and return the address of an object in a system pointer.
- Addressing an object indirectly through an object in a library: An object addressed through a library contains system pointer(s) to other objects not usually addressable through the library. This scheme is used by composite objects; they are a collection of objects that are logically tied together to provide some function (for example, libraries are composite objects consisting of the context, which is addressable through the machine context, the object description storage objects, and a programming change object).
- Addressing objects through the owning-user profile: To assist in this, vertical microcode provides a function by which any objects that are unowned can be found (Reclaim instruction). Using the output from this instruction and a list of owned objects for all profiles, any object in the machine can be addressed. This scheme is used by the reclaim storage utility to clean up objects that are not properly addressable by the first two schemes.

Those objects that should be addressed through a context and are not will be placed back into the appropriate context or a special reclaim context. Non-context addressable pieces of composite objects will be relinked to either the piece that is addressable or to a reconstructed piece, or will be deleted. These pieces are referred to as dangling.

After running the CPF reclaim utility, all objects created prior to the last IMPL (internal microprogram load) should be addressable through the first two schemes.

Reclaim Overview

Figure RC-1 and the following text describe the flow of control when running the reclaim function. All subsystems must be terminated; this restriction allows the function to avoid any of the normal locking problems.

1 The reclaim driver QRCLAIM gets control, checks to determine if the system is quiesced, and if commitment control is active; the message handler is called to inform the user if the subsystem is not quiesced, and if commitment control is active. The data base and librarian components are called to perform any recovery actions. The librarian component is called to return a list of the libraries on the system, and these are checked for damage. Upon encountering a damaged library (context), the message handler is called to inform the user. QLIRCLIB is invoked to check all of the objects that comprise a library for damage. If damage is detected, the object will be deleted and re-created. QLIRCLIB will also rebuild any library whose associated space is found damaged. The objects required by the reclaim function are created next. They include a communication object which is passed between the modules, and they contain object processing information and pointers to other objects. Also created is the QRCL library. During all stages of processing, the user is informed with status messages on the progress of the function.

2 QRCLAIM calls QRCALOWN, passing a pointer to the communication object. This module calls the librarian component to return a list of the profiles on the system. These are checked for damage, and the user is informed if QSYS, QDBSHR, QSNADS, QDOC, QSPL, or QSECOFR are damaged. If any other profile is damaged, it is deleted, and the operator message queue has a message placed on it for the profile deletion. All profiles are now undamaged. The vertical microcode reclaim function is called to return a list of ownerless objects. This list is processed and a default owner is given to any object without an owner. Additionally, duplicate profiles and libraries, logical unit descriptions, control unit descriptions, and line descriptions that are not addressable through the machine context are deleted. Control is then returned to QRCLAIM.

3 QRCLAIM processes the list of valid profiles that was materialized in QRCALOWN. The current profile pointer is stored in the communication object. Control is then passed to QRCSEPOB. module.

4 QRCSEPOB materializes the owned objects for the current profile. Each object is then processed and placed into one of five classifications. Objects that are formats or directories are handled by entering information on them into a list. The primary part of a composite object is handled by passing control to QRCOMPST. Information on the secondary pieces of a composite object is saved in an index for processing. Objects with types that are invalid to vertical microcode or subtypes that are invalid to CPF are handled by transferring the invalid-typed objects to the security officer and deleting the invalid-subtype objects. All other objects are processed in QRCSIMPL. After all owned objects for a profile are handled, control is then returned to QRCLAIM.

5 When QRCSIMPL gets control from QRCSEPOB, the object is verified as being in the proper library. Objects previously in a temporary library are deleted, and any whose context cannot be found or are duplicates of objects in a library are moved to QRCL. This excludes internal CPF objects and objects owned by QSYS, which are considered as IBM-supplied system objects. These are deleted when encountered as duplicates.

6 When QRCOMPST gets control from QRCSEPOB, QLIRCLIB is called if the object is a library, to see if the objects that comprise this library have been checked for damage. If not, they will be checked and, if found damaged, the object will be deleted and re-created. If the associated space is damaged, the library will be rebuilt. Next, the object is verified as being in the proper library. Objects previously in a temporary library are deleted, and any whose context cannot be found or are duplicates of objects in a library are moved to QRCL. The secondary pieces of the object are then processed, and information on the pieces is stored into an index for later processing. For data base files, QDBRCLMA is invoked to return a list of secondary pieces. As secondary pieces are processed, the ownership and authority is made consistent with the primary piece.

7 Objects are inserted into the QRCL library by calling QRCINSRT. This module inserts the object that was passed and renames the object to an alias if a duplicate object already exists in QRCL. The OIR text description is updated to inform the user of the original name and library of the object that was inserted.

8 After all profiles are processed, QRCLENUP is invoked to handle the secondary pieces of composite objects having no associated primaries. The list of these objects is obtained from the index containing secondary piece information. Those pieces of a library or subsystem description are deleted. For files, QDBRCLMB is called to create file structures for those pieces still containing lost user data. After all secondary pieces are handled, QDBRCLMC is called to clean up formats and directories no longer containing valid information. Extraneous data base recovery objects are also deleted. Control is returned to QRCLAIM.

9 All objects on the system should be properly addressable. QLIVLOIR is invoked for each library to remove object description information for those objects no longer in the library. After control is returned to QRCLAIM, the module deletes the internal objects created and also deletes the QRCL library if it is empty. The message handler is called to output completion information and the function is terminated.

10 If a job is being serviced during execution of this function, a series of dumps will be taken based on calls from the various modules to QRCDEBUG. The profiles and libraries on the system are listed. Any object that is ownerless is listed with who the default owner was. Objects moved to QRCL will result in a call to output the information. Secondary pieces that are not attached to a primary are listed prior to their deletion or recreation back into valid files. Any object being deleted from the system is listed. This information is also listed in the system operator's message queue, for those objects known to the user (external). QRCDEBUG also outputs information for those objects internal to the system of which the user has no knowledge.

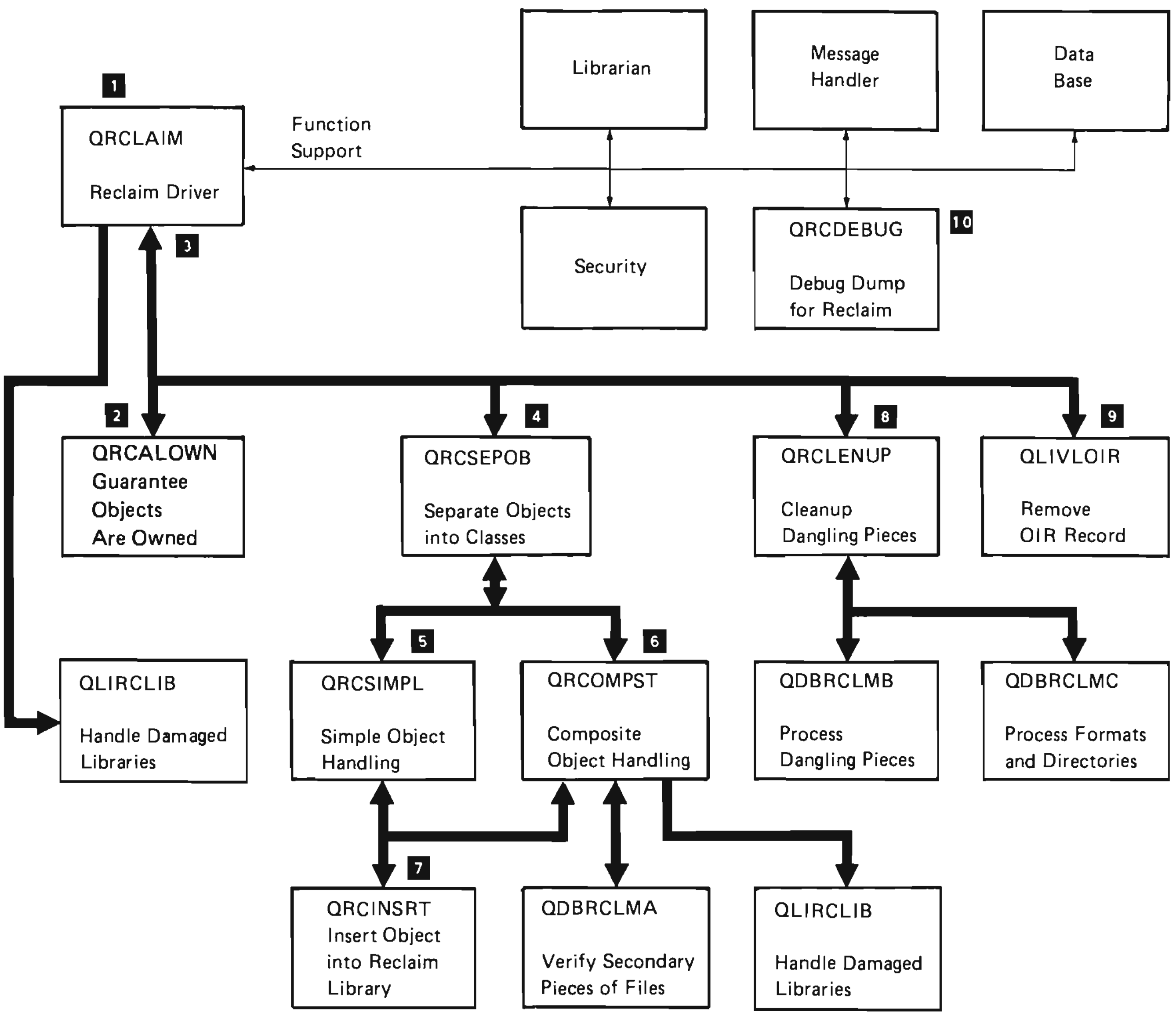


Figure RC-1. Reclaim Overview

GENERAL OVERVIEW OF DAMAGE NOTIFICATION

Damage Notification Modules

The damage notification portion of this component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

Damage notification modules may fail due to the conditions they are running in; however, a failure in these programs is handled by the calling program and will have no further effect on existing processes.

-->QRCIMPLN–Logging of Damaged Objects During IMPL to the History Log: This module receives control from the work control component during an IMPL. QRCIMPLN searches the object recovery list for damaged object entries placed on it by vertical microcode. A CPF message is created for each entry and sent to the history log. QRCIMPLN places message CPF8197 on the system operator message queue if it has sent any messages to the history log. If there is not enough auxiliary storage to send all of the damaged object messages, QRCIMPLN will send message CPF8196 to the system operator message queue.

-->QRCDMGLG–Logging of Damaged Objects to the History Log: This module handles machine interface event hex 0017-0401 for the event monitor, which is established in the system arbiter. QRCDMGLG retrieves the event data for the damage set event, and using this data creates a CPF message. QRCDMGLG places the message on the system operator message queue. It also places message CPF8198 on the system operator message queue. This automatically places the messages on the system history log. It returns to the system arbiter.

-->QRCPDMGL–Logging of Partially Damaged Objects to the History Log: This module receives control from the switched line component to handle the hex 0017 0801 event for all objects excluding I/O descriptions (logical unit, control unit, and line) using the data passed. A CPF message is created using the data passed via a parameter. QRCPDMGL places the message on the system operator message queue. It also places message CPF8198 on the system operator message queue. This automatically places the messages on the system history log. Control is returned to the switched line component.

-->QRCDMGNT–Default Program for Damage Notification: This module acts as the default program for any unhandled MCH1604 escape messages. It is available to other CPF programs for use in damage notification. When called, QRCDMGNT retrieves the exception data from the message queue. It determines what type of message to send, builds the correct corresponding message data, and sends this message to the message queue of the program that received the original damage exception.

-->QRCPDMGN–Default Program for Partial Damage Notification: This module acts as the default program for any unhandled MCH1668 escape messages. It is available to other CPF programs for use in damage notification. When called, QRCPDMGN retrieves the exception data from the message queue. It determines what type of message to send, builds the correct corresponding message data, and sends this message to the message queue of the program that received the original damage exception.

The following programs are called by QRCIMPLN, QRCDMGLG, QRCDMGNT, QRCPDMGL, and QRCPDMGN:

QWDDMGNT–Subsystem Description Damage Notification Program: This module is called by the damage notification programs to determine the subsystem description of which the damaged object is a piece. QWDDMGNT creates the message data and indicates which message to send. It returns to the calling program.

QLIDMGNT–Librarian Damage Notification Program: This module is called by the damage notification programs to determine the library of which the object is a part. QLIDMGNT then creates the message data and indicates which message to send. It returns to the calling program.

QDBDMGMB–Data Base File-Member Damage Notification Program: This module is called by the damage notification programs to determine which file member has been damaged. It also determines the type of piece (cursor, data space, or data space index for example). QDBDMGMB creates the message data and indicates the message to be sent. It returns to the calling program.

QDMDMGNT–Data Management File Damage Notification Program: This module is called by the damage notification programs to determine the file and the type of file that is damaged. QDMDMGNT creates the message data and indicates the message to be sent. It returns to the calling program.

QMHDMGNT–Message Queue Damage Notification Program: This module is called by the damage notification programs to determine the type of message queue that is damaged. QMHDMGNT creates the message to be sent. It returns to the calling program.

QSPDMGNT–Spool Control Block Damage Notification Program: This module is called by the damage notification programs to determine the job that is using the damaged spool control block. QSPDMGNT creates the message data and indicates the message to be sent. It returns to the calling program.

QJODMGNT–Journal Receiver Damage Notification Program: This module is called by the damage notification programs to determine full or partial damage of the damaged journal receivers. QJODMGNT creates the message data and indicates the message to be sent. It returns to the calling program.

QWCDMGNT–Local Data Area Damage Notification Program: This module is called by the damage notification programs to determine the name of the job that is using the damaged local data area. QWCDMGNT creates the message data and indicates the message to be sent. It returns to the calling program.

QDFDMGNT–Save File Damage Notification: This module is called by the damage notification programs to determine which save file is damaged. QDFDMGNT controls the message to be sent. It returns to the calling program.

Damage Notification Function

The CPF damage notification and logging functions handle the entry of damaged objects placed on the object recovery list if the system went through directory recovery during IMPL, and the signaling of machine interface damage events and exceptions. This results in the following facilities:

- During IMPL, a damage notification module searches the object recovery list for entries placed on it by vertical microcode. This module creates and sends a message to the history log for each damaged object entry it encounters.
- The damaged object event is signaled when the machine interface object is marked as damaged by the machine. A damage notification module handles this event and sends two damage messages to the system operator message queue and two damage messages to the history log.
- When the machine signals an object-damaged exception that is not handled by the proper program in a process, a module is invoked to interrogate and notify the system user of the damaged object (internal or external).

Concepts

Most CPF components are not expecting to find damaged objects. Therefore, most of the machine-signaled damage exceptions go unmonitored. This causes the message handling component to issue the function check escape message to the process. This usually causes the process to forego normal completion of the function being performed. From the information returned with the machine exception, an operator or user may be unable to determine which object is damaged or what corrective actions to take.

- Some CPF objects are composite, that is, composed of more than one machine interface object. Only the primary machine interface object of the composite is addressed through a context and only the primary object must follow naming standards. The end user may not be aware of the use of a composite object. Returning the name of any piece of the composite would only confuse the user.

- The machine interface exception data contains a pointer to the damaged object. From the pointer, the name of the object, its type and subtype codes, the name of the library that contains the object, and the owning user profile can be found.

- Many of the objects are known only to CPF. These objects can also become damaged and cause processes in the system to behave in unusual ways. The user must be notified in these cases to determine the proper corrective procedures.

- The user must be able to relate a particular object's damage to the means of correcting the damage.

Assumptions

If any of the assumptions are not met, it is possible that the damage notification modules will not be invoked properly. Therefore, the results in these cases are unpredictable.

Exception Handling Program Assumptions

- The input parameters are correct regardless of the method of invocation; that is, there is always a valid message queue and it contains the message identified on the invocation of the program.

- The message queue that contains the message is not the damaged object.

- The damaged object is not the message file being used for message handling in the process.

- The damage is not to any module that may be involved in determining the damaged object or sending the resulting messages.

- The resulting messages exist in the specified CPF message file and contain the proper data.

Event Handling Program Assumptions

- The program only executes when the machine interface damage set event occurs.

- The system history log or system operator message queue is not the damaged object.

- The messages appear on the appropriate CPF message file and contain the proper text and data.

- The same messages are placed on the history log as appear for damage notification.

Damage Notification During IMPL Overview

Figure RC-2 shows an overview of the logging of damage notification during IMPL.

1. During an IMPL, QWCISCFR calls QRCIMPLN.

2. When QRCIMPLN is called, it searches the object recovery list for physically damaged object entries placed on it by vertical microcode during directory recovery. QRCIMPLN then determines which message to send and then sends it to the history log. If QRCIMPLN has sent any messages to the history log, it also sends message CPF8197 to the system operator message queue. If there is not enough auxiliary storage to send all of the damaged object messages, QRCIMPLN will send message CPF8196 to the system operator message queue.

3. QRCIMPLN calls QDBRCIPS to handle data base recovery.

4. After QDBRCIPS has completed data base recovery, it returns control to QRCIMPLN.

5. QRCIMPLN calls QTNIPL to handle commitment control initial program load recovery.

6. After QTNIPL completes recovery, it returns control to QRCIMPLN.

7. QRCIMPLN then returns control to QWSISCFR.

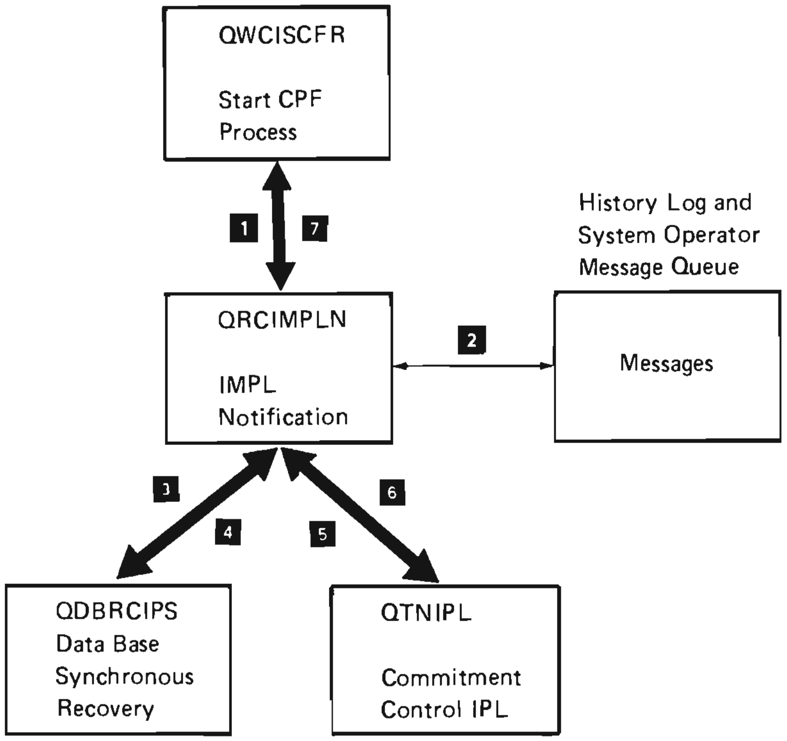


Figure RC-2. Damage Notification During IMPL Overview

Damage Notification Overview

Figure RC-3 shows an overview of damage notification for damage exceptions.

1 A process is executing all kinds of instructions and suddenly incurs a damaged machine interface object. The machine signals a hard damage exception (hex 1004). The same would hold for partial damage except that control would pass to QRCPDMGN instead of QRCDMGNT.

2 The process does not listen for any machine interface exceptions directly, so the exception gets to the PDEH (process default exception handler). PDEH places the exception data in the message queue and signals an MCH1604 escape message to the proper invocation of the process.

3 If this exception is not handled by the program, and depending on the module, message severity, and the setting of the service log indicator in the message description, PDEH may send the unhandled escape message to the service log.

4 PDEH invokes the default program identified in the message file for this unhandled exception, in this case QRCDMGNT.

5 QRCDMGNT obtains addressability to the damaged object and determines the user or system object that has been damaged as a result. Then QRCDMGNT sends an escape message to the message queue of the program that was originally signaled for the damage exception. This places another escape message on the message queue, a message that defines the damage by the use of symbolic names. The user can listen explicitly for this CPF message.

6 If the user does not listen for the new message, PDEH signals a function check back through his program.

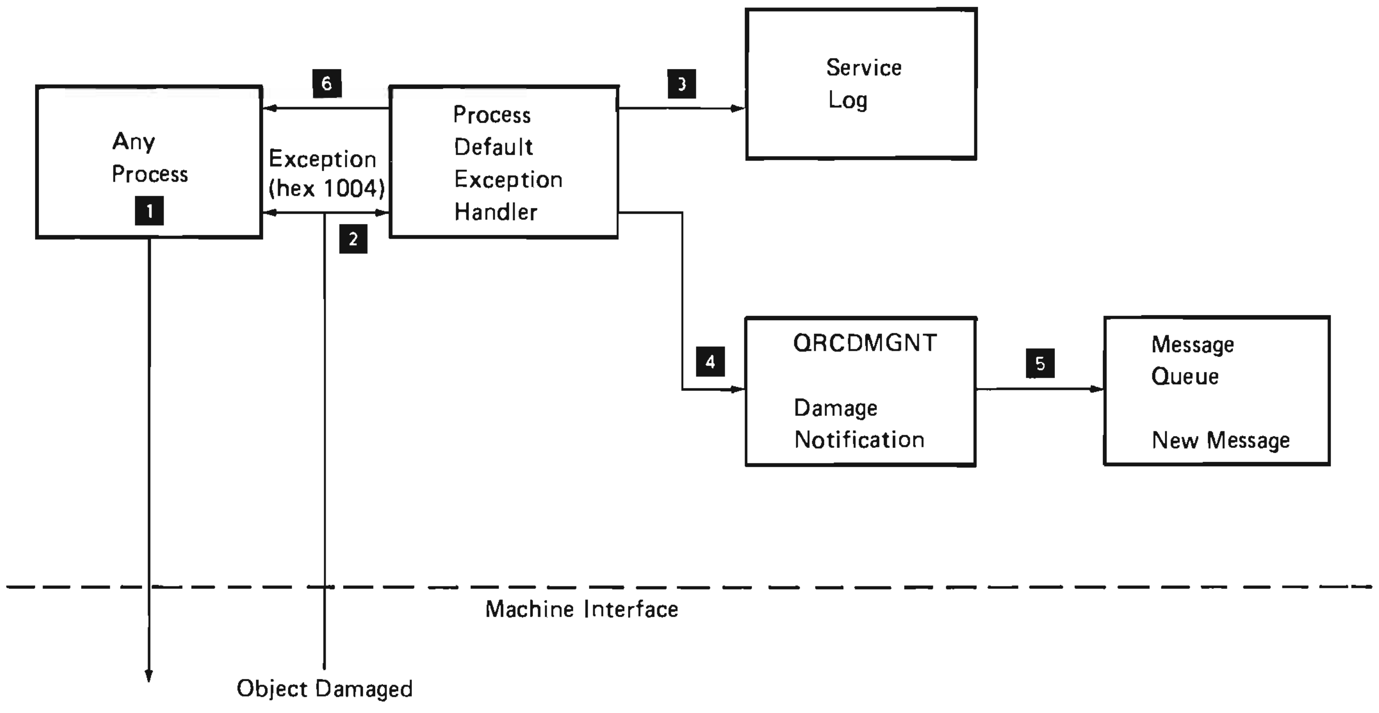


Figure RC-3. Default Program for Damage Notification Overview

Logging of Damaged Objects on the History Log Overview

Figure RC-4 shows an overview of the logging of damaged objects on the history log.

1 As the system is brought up and the system arbiter process is established, the system arbiter starts monitoring for the machine interface event for the system object damage set event. Once the arbiter establishes QRCDMGLG as the program to handle this event, QRCDMGLG will be executed. In the case of partial damage events, the arbiter establishes QRCPDMGL as the handling program to be executed.

2 When QRCDMGLG is invoked, it retrieves the event-related data and use it to obtain addressability to the damaged object. QRCDMGLG then determines which message to send to the system operator and system history log. It also sends message CPF8198 to the system operator and system history log.

After sending the messages, the program returns to the system arbiter.

Partial damage events (hex 0017 0801) are handled similarly; the damage notification module is QRCPDMGL, and QSWERP, the switched line module, is called between the system arbiter and QRCDMGLG. QSWERP handles the partial damage event for logical unit, control unit, and line descriptions.

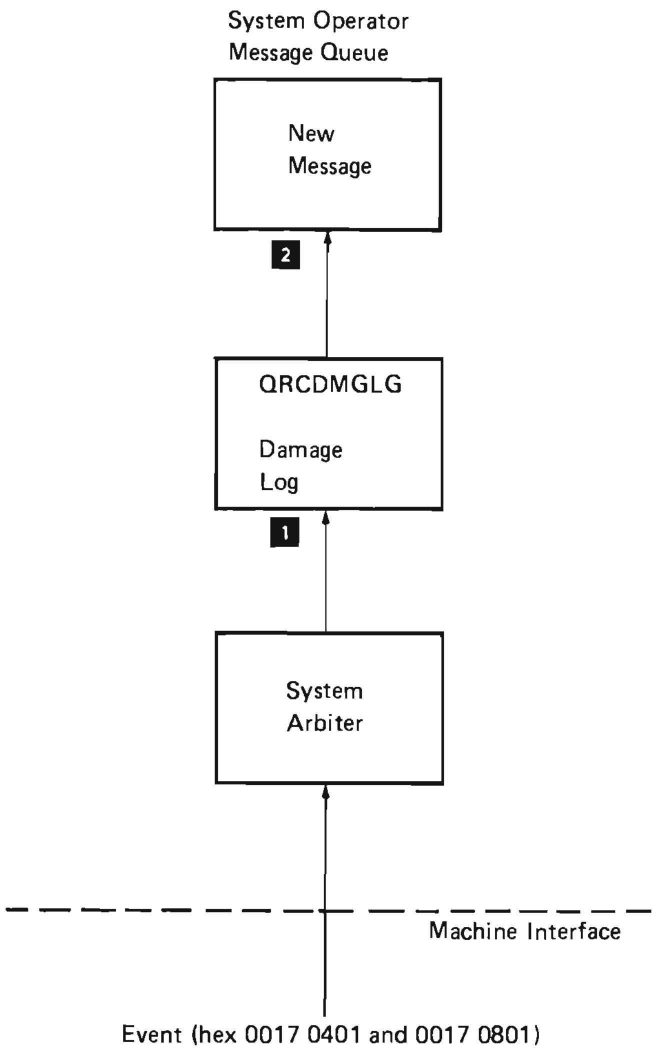


Figure RC-4. Damaged Objects on the History Log Overview

Special Case Programs Overview

Figure RC-5 shows an overview of special case programs.

1 Each of the special case modules is invoked in the same way. One of the damage notification programs (QRCIMPLN, QRCDMGLG, QRCPDMGL, QRCDMGNT, or QRCPDMGN) is notified of damage.

2 It calls one of the special case programs to determine the proper message and data to send.

When the program has determined which message and data to send, it returns these values to the calling routine and returns in the normal way.

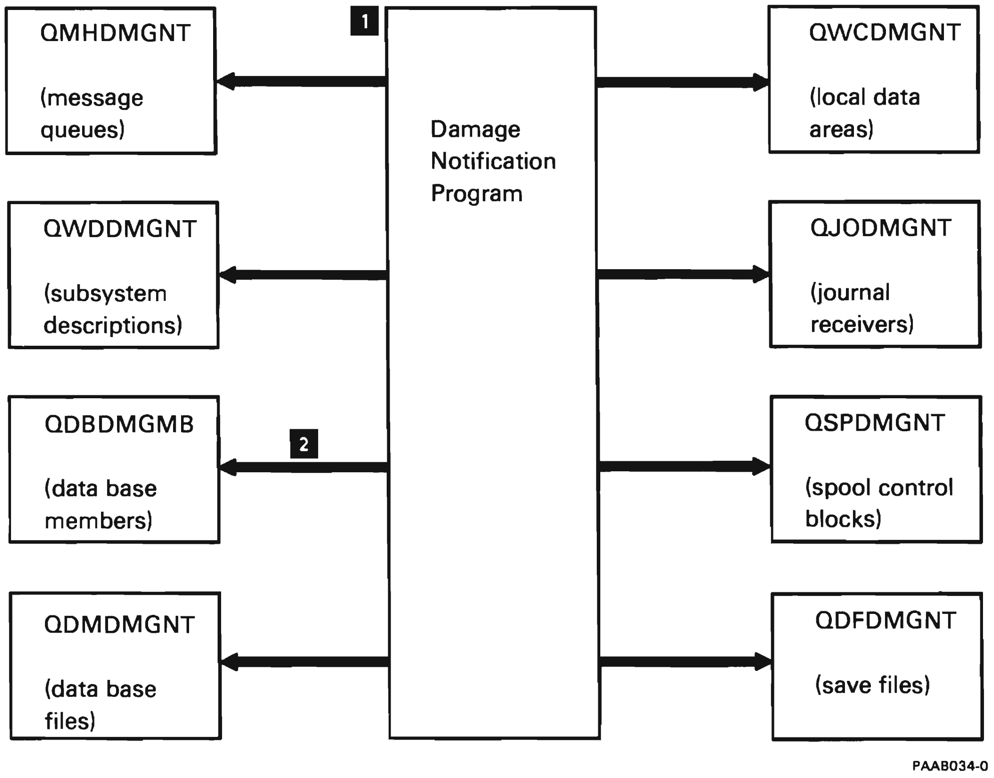


Figure RC-5. Special Case Programs Overview

INTRODUCTION

The service component of the CPF (control program facility) provides aids and tools to assist the user (primarily the service personnel) in problem determination, analysis, reporting, and repair of system program troubles within the CPF. An interface to the internal machine product diagnostics and service aids is also provided by the concurrent service monitor component.

The service aids and tools are accessible through CL commands and an internal macro interface. They are:

- Alert messages
- Dumps
- Trace
- Interjob servicing
- Programming changes
- Programming patches
- APAR (authorized program analysis report) data preparation
- Programming change log
- Service log
- Internal service facility
- System verification facilities

GENERAL OVERVIEW

Service Modules

The service component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QSCAEVTH–Alert Event Handler: This module formats and sends an alert to the network host.

-->QSCAPAR–Prepare APAR (PRPAPAR)[1]: This module saves objects needed for the resolution of CPF programming problems on the save/restore device. It builds a list of objects to be saved along with the system logs and programs, data base files, and spooled files that were specified on the Prepare APAR command.

> QSCCOPY–Copy APAR Files: This module performs the copy function which copies spool files into data base files.

-->QSCAPC–Apply Programming Change (APYPGMCHG)[1]: This module performs the validity checking and program change prerequisite checking prior to calling QSCARPC to apply the program change.

> QSCARPC–Apply/Remove Programming Change: This module is used by all of the programming change and patch functions. QSCARPC performs the apply program change or remove program change function (if programming change exit programs exist, they are called after the programming change is applied or removed). It is called by QSCAPC, QSCLPC, QSCPP, QSCRPC, or QSCCPFI.

[1]This module is a CPP (command processing program).

QSCAUTCK–Authority Checker: This module checks if the user is authorized to perform programming changes in the library chosen.

-->QSCCLNPT–Clean Print Train (CLNPRT)[1]: With the print train ribbon removed on the 3203, this module causes a ripple pattern to be imprinted on a special form, which cleans the print train.

-->QSCCCNPA–Change CSNAP Attributes (CHGCNPA)[1]: This module changes the CSNAP short term statistics attributes which are set in the system.

-->QSCCPFI–Start CPF Interface: This module is invoked during the start CPF process. It lets the user apply or remove deferred programming changes. A display, containing the modifiable deferred programming changes, is presented to the user. The user can then modify the status of the program changes. QSCARPC is called to perform any changes.

-->QSCCPYLG–List Error Log (LSTERRLOG)[1]: This module causes error log information to be written to a spooled printer file.

-->QSCCRTPG–Encapsulate Program: This module is invoked to create a program from the materialized program template in a data base file.

-->QSCDCNPA–Display CSNAP Attributes (DSPCNPA)[1]: This module displays or lists the CSNAP short term statistics attributes set in the system.

-->QSCDJI–Dump Job Internal (DMPJOBINT)[1]: This module gets an SCO (service communications object), by creating one or resolving to one if it already exists, to determine if a different job is being serviced. If a different job is being serviced, a message is sent to the other job and QSCDH is called. If a different job is *not* being serviced, control is transferred to QSCGJI to produce the dump.

-->QSCGJI–Get Internal Job Dump: This module performs an internal job dump and sends a message to the requestor containing the dump identifier.

-->QSCDJOB–Dump Job (DMPJOB)[1]: This module gets an SCO by creating one or resolving to one if it already exists, and it copies the command parameters to the SCO of the job being serviced. If another job is being serviced, a message is sent to the other job and QSCDH is called. If a different job is *not* being serviced, control is transferred to QSCGJOB to produce the dump.

QSCDH–Service Data Handler: This module opens and closes the printer device file when a job is in service mode. It also sends, to the printer dump, data that was sent to a servicing job by its serviced job.

QSCGJOB–Get Job Storage Dump: This module opens and closes the printer device file, sends out dump heading lines, creates a common materialization space, and then calls QSCJSDMP.

-->QSCDOBJ–Dump Object (DMPOBJ)[1]: An object to be dumped can be selected by qualified object name, and object type. This module gets an SCO by creating one or resolving to one if it already exists, and copies the command parameters to the SCO of the job being serviced. If another job is being serviced, a message is sent to that job and QSCDH is called. If another job is *not* being serviced, control is transferred to QSCGSO to produce the dump.

QSCDH–Service Data Handler: This modules opens and closes the printer device file when a job is in service mode. It also sends, to the printer dump, data that was sent to a servicing job by its serviced job.

QSCGSO–Get System Objects: This module gets system objects for the Dump System Object and Dump Object commands. QSCGSO opens and closes the printer device file, outputs dump heading lines, creates a common materialization space, and calls QSCOBJDM to dump the specified objects.

-->QSCDMPPS–Dump Pointer in a Space: This module dumps materialization of the pointer in a space for the ?DMPDTA macro.

[1]This module is a CPP (command processing program).

-->QSCDSO–Dump System Object (DMPSYSOBJ)[1]: A system object to be dumped can be selected by context, generic name, and type. The object to be dumped can also be selected by specifying a base object and a list of offsets to pointers which chain to the object to be dumped. A portion of the object's associated space can be selected instead of a dump of the full object. This module gets an SCO by creating one or resolving to one if it already exists, and copies the command parameters to the SCO of the job being serviced. If a different job is being serviced, a message is sent to it and QSCDH is called. If a different job is *not* being serviced, control is transferred to QSCGSO to produce the dump.

QSCDH–Service Data Handler: This module opens and closes the printer device file when a job is in service mode. It also sends, to the printer dump, data that was sent to a servicing job by its serviced job.

QSCGSO–Get System Objects: This module gets system objects for the Dump System Object and Dump Object commands. QSCGSO opens and closes the printer device file, outputs dump heading lines, creates a common materialization space, and calls QSCOBJDM to dump the specified objects.

-->QSCDSS–Display Service Status (DSPSRVSTS)[1]: This module displays the names of the serviced and servicing jobs, and information about the CPF trace if it is active.

-->QSCEND–End Service (ENDSRV)[1]: This module takes a job out of the service mode by destroying the SCO, the service queues, and resetting the flags in the work control block.

-->QSCFSODD–Format and Send Out Dump Data: This module formats and sends out lines of hexadecimal data.

-->QSCINTDD–Internal Dump Data: This module performs an internal job dump for the ?DMPDTA macro.

-->QSCJSDMP–Dump Job Storage Area: This module dumps the PASA, PSSA, object mapping table for observable programs, job structure objects, and all spaces addressed by space pointers in the PASA and PSSA for a specific job.

-->QSCLCNPD–List CSNAP Data (LSTCNPDTA)[1]: This module lists the CSNAP short term data in the system.

-->QSCLCNPH–List CSNAP History (LSTCNPHST)[1]: This module lists the CSNAP history data in the system.

-->QSCLID–List Internal Data (LSTINTDTA)[1]: This module lists the requested internal machine data areas.

-->QSCLCPR–Load Programming Change Authority Checking (LODPGMCHG)[1]: This module checks if the user is authorized to load programming changes to the library specified.

QSCLPC–Load Programming Change: Programming changes are read into the system from the program change media by a restore operation requested by this module. It is called by QSCLCPR.

-->QSCMATPG–Materialize Program: This module is called to materialize a program, and place the materialized program template in a data base file.

-->QSCMMCTX–Materialize and Dump Machine Context: This module materializes and dumps the attributes of the machine context, including the identifications of all objects addressed through the context.

[1]This module is a CPP (command processing program).

-->QSCOBJDM–Object Dump: This module routes a dump object or dump system object request to the proper materialize and dump routine. QSCOBJDM also outputs the object heading, dumps the object's associated space (if it has one) and dumps the object's OIR (object information repository) data.

QSCMAG–Materialize and Dump Access Group: This module dumps the attributes of an access group and the identification of all objects contained in that access group.

QSCMCB–Materialize and Dump Commitment Block: This module materializes and dumps the attributes of a commitment block, including the commitment block status, the commitment description, and objects associated with the commitment block.

QSCMCD–Materialize and Dump Controller Description: This module materializes and dumps the attributes of a controller description, including backward and forward object lists and the network description candidate list if one is present.

QSCMCTX–Materialize and Dump Context: This module materializes and dumps the attributes of a context and the identification of the objects addressed through the context.

QSCMCUR–Materialize and Dump Cursor: This module materializes and dumps the attributes and operational statistics of a cursor, including the identification of all data space indexes that the cursor is over.

QSCMDMPS–Materialize and Dump Dump Space: This module materializes and dumps the attributes and size values for a dump space.

QSCMDS–Materialize and Dump Data Space: This module materializes and dumps the attributes and operational statistics of a data space, including the identification of the data space index if there is one over the data space.

QSCMDSI–Materialize and Dump Data Space Index: This module materializes and dumps the attributes and operational statistics of a data space index, including the identification of any data spaces addressed by the index.

QSCMIDX–Materialize and Dump Index: This module materializes and dumps the attributes and entries in an independent index.

QSCMJPRT–Materialize And Dump Journal Port: This module materializes and du the attributes of a journal port, including attached journal spaces and objects being journaled.

QSCMJSPC–Materialize and Dump Journal Space: This module materializes and dumps the attributes of a journal space, including the associated journal port.

QSCMLUD–Materialize and Dump Logical Unit Description: This module materializes and dumps the attributes of the logical unit description including the identification of the object addressed by the forward object pointer.

QSCMND–Materialize and Dump Network Description: This module materializes and dumps the attributes of a network description, including the identifications of objects in the backwards object list.

QSCMPGM–Materialize and Dump Program: This module materializes and dumps the program template. Only the program components that can be materialized as observable will be dumped.

QSCMQ–Materialize and Dump Queue: This module materializes and dumps the attributes of a machine interface queue.

QSCMSP–Materialize and Dump Space: This module materializes and dumps the attributes of a system space.

QSCMUP–Materialize and Dump User Profile: This module materializes and dumps a user profile and its list of authorized objects.

-->QSCSPCDM–Space Dump Routine: This module dumps a space or part of a space for dump system object.

-->QSCPCSYS–Display Programming Changes on the System (DSPPGMCHG)[1]: This module gathers information on programming changes, then produces programming change and patch status displays and listings.

QSCDSPPC–Display Detailed Information for a Programming Change: This module creates the detailed information display or listing for a programming change.

-->QSCOIR–Retrieve Service OIR: This module is used by application development routines to retrieve the service OIR (object information repository) data.

-->QSCONEOB–Single Object Dump: This module provides a dump of a single machine interface object. The module is called by QDMROUTE to dump a single non-standard object.

-->QSCPDPC–Start Problem Determination Procedure (STRPDP)[1]: This module executes the specified problem determination procedure.

QSCPDPXH–Problem Determination Procedure Exception Handler: This module handles the service function terminated exception (CPF7206).

-->QSCPP–Patch Program (PCHPGM)[1]: This module is used to patch a program and then re-encapsulates the program to create a new program as a local programming change.

-->QSCPUTR–Service Open/Close: This module opens and closes the printer device file for dump operations.

-->QSCRH–Service Request Handler: This module handles the service request messages sent between jobs engaged in interjob servicing.

-->QSCRPC–Remove Programming Change (RMVPGMCHG)[1]: This module checks that the program change being removed is not a prerequisite for some other installed program change, and then calls QSCARPC to remove the programming change.

QSCAUTCK–Authority Checker: This module checks if the user is authorized to perform programming changes in the library chosen.

-->QSCSVJOB–Service Job (SRVJOB)[1]: This module places a specified job in service mode.

-->QSCTEVTH–Trace Job Event Handler: This module builds the trace records.

-->QSCTI–Trace Internal (TRCINT)[1]: This module activates or deactivates the specified internal trace.

-->QSCTJOB–Trace Job (TRCJOB)[1]: This module activates the trace function for the specified job.

QSCPTREC–Trace Record Output Formatter: This module formats trace records and then prints th formatted output.

QSCTJOBR–Trace Job Remote Setup: This module initiates the CPF trace in the serviced job as a result of the Trace Job command and the initialization performed by QSCTJOB.

-->QSCVFYPT–Verify Printer (VFYPRT)[1]: This module processes the Verify Printer command.

-->QSCWMINT–Service/Work Management Interface: This module is called by the process termination and process initiation functions of work management to perform necessary service functions, including set trace on, set trace off, and terminate service mode.

Alert Messages

All messages to the history log (QHST) that have an alert ID other than *NONE are forwarded to the network host when alert messages are being processed. The message handler component invokes QSCAEVTH to send the alert messages to the network host. The following command is used to initiate alert messages processing:

- Change Network Attributes (CHGNETA): This command initiates alert processing by changing the network attributes.

Dumps

Dump commands provide hexadecimal dumps of individual system objects, objects that make up a job structure, and combinations of job structure and related system objects. An internal macro interface is also available to the system default escape message handler to take dumps when an unmonitored escape message is sent. Dumps are written to a spooled print file that can be held after printing for submission with an APAR. The following commands are used to perform dumps:

- Dump Object (DMPOBJ): This command is used to dump an external object that is stored in a library.
- Dump System Object (DMPSYSOBJ): This command is used to dump any machine interface object that is stored in a context or can be indirectly addressed through an object stored in a context.
- Dump Job (DMPJOB): This command is used to dump the job structure and related objects.

Trace

Invocations and invocation terminations of machine interface programs can be traced. The Trace Job (TRCJOB) command is used to turn trace on or off, to specify the trace table storage area size, the action to take if the trace table becomes full (wrap or stop), the type of tracing to be done, and, optionally, the name of a user exit program to be invoked for each trace record generated. The trace can also be cancelled without printing the trace records.

In addition to the flow trace, there is a data trace that generates specific program data instead of the normal trace record.

A flow trace record contains the following information:

- Time stamp of when the trace record was generated
- Trace record sequence number
- Record type–call, transfer control, event handler invocation, external exception handler invocation, invocation due to an invocation exit routine, return, invocation terminated, internal or branchpoint exception handler invocation, invocation terminated for intervening invocation, internal exception handler return, external exception handler return, process termination phase terminated, and process terminated for unhandled exceptions.
- Program name–the name of the program
- The name of the library addressing the program
- Instruction number of the last entry to the program
- Instruction number of the exit from the program
- Invocation number

A data trace record contains the following:

- Time stamp of when the trace record was generated
- Trace record sequence number
- Record type-data
- Trace point data

A user exit program gives a trace user an opportunity to perform additional functions to those provided by the CPF trace. For example, the exit program could suppress a trace record by blanking it out, or it could invoke the dump commands.

- CPU time used
- Non-data-base page reads
- Data base page reads
- Pages written
- Transitions to wait state

Interjob Servicing

Normally service dump and trace commands apply to the job in which they are entered; however, one job may service another job. When a job is in service mode (servicing another job), dump and trace commands entered in the job cause dumps and traces to be performed in the serviced job. The following commands are used to perform interjob servicing:

- Service Job (SRVJOB): This command specifies the job to be serviced and establishes service mode.
- End Service (ENDSRV): This command ends service mode.
- Display Service Status (DSPSRVSTS): This command displays the service status of a job-whether servicing another job or being serviced by another job, and the status of the job being traced.

The following commands are affected by service mode:

- Dump Job (DMPJOB)
- Dump Job Internal (DMPJOBINT)
- Dump Object (DMPOBJ)
- Dump System Object (DMPSYSOBJ)
- Trace Job (TRCJOB)
- Trace Internal Data (TRCINT)

Programming Changes

Programming changes are created in an IBM laboratory and placed on a diskette for distribution. These programming changes are in response to an APAR and consist of those system objects that must replace the failing system objects to correct the programming problem.

Programming changes are installed with the Load Program Change (LODPGMCHG) command. All immediate programming changes can be applied temporarily with the Apply Program Change (APYPGMCHG) command. If a temporarily applied program change does not solve the problem, it can be removed with the Remove Program Change (RMVPGMCHG) command. If the user is confident that the programming change is going to solve the problem, the programming change can be permanently applied. Once a program change has been permanently applied to the system, it cannot be removed and becomes a permanent part of the system.

Almost all CPF programming changes are deferred, in special situations only can CPF programming changes be temporarily applied to the system as it is running. There are, however, some special programming changes, although loaded into the system with the others, that must be applied when the system is not fully operating. These deferred program changes can be applied at the start CPF process time. A display is presented to the operator to allow the selection of those program changes. Once these deferred changes are permanently applied, they cannot be removed.

Programming changes remain valid when a save system operation takes place and a corresponding restore takes place.

Program Patches

Emergency repairs to a program can be made by patching the program. The Patch Program (PCHPGM) command is used to create a patched copy of a program. The resulting patched program is handled by service as a locally generated program change. If the job is an interactive job, a display can be requested that shows selected portions of the program template. Patch data can then be entered on the display screen. If the job is a batch job, patch data can be entered through the QPCHSRC source file. Patch data can be verified by checksum checking. Verification is specified by specifying the CHKSUM parameter on the Patch Program command.

A log record is built for each program patch. It is printed and logged to the CPF service log.

The status and attributes of all programming changes and patches on the system can be displayed with the Display Programming Changes (DSPPGMCHG) command.

Programming Change Log

The programming change log is a system log used to track the loading, application, and removal of programming changes and program patches. Each entry is a message that was sent to QCHG. A message is sent to the log for the following reasons:

- Program changes loaded in the system
- Programs patched
- Program changes or patches applied temporarily
- Program changes or patches applied permanently
- Program changes or patches removed temporarily
- Program changes or patches removed permanently
- Messages sent by the system operator or service personnel to explain the action taken

APAR Data Preparation

Information pertaining to a CPF programming problem can be saved on a save/restore device volume for later analysis by laboratory personnel. This information can include data base files, programs suspected of being in error, an MTR (machine trouble report) and spooled print files that contain dump and trace output as well as the user output necessary to show the type of program failure. The Prepare APAR (PRPAPAR) command is used to gather and save this information.

Internal Service Facilities

The internal service facility provides aids and tools to service the internal machine product. The commands used to provide this support are:

- List Error Log Data (LSTERRLOG): Information about machine checks, device errors, and volume statistical data is recorded by the machine in a log that is not normally accessible above the machine interface. This command allows selected log data to be listed to a special printer file.
- Trace Internal (TRCINT): This command allows tracing of various activities below the machine interface.
- List Internal Data (LSTINTDTA): This command prints the contents of internal machine data areas.
- Dump Job Internal (DMPJOBINT): This command dumps the machine data structures related to the machine process in which the current job is executing.
- Change CSNAP Attributes (CHGCNPA): This command allows the user to change the CSNAP short term statistics attributes in the system.
- Display CSNAP Attributes (DSPCNPA): This command displays the CSNAP short term statistics attributes which are set in the system.
- List CSNAP Data (LSTCNPDTA): This command lists the CSNAP data in the system.
- List CSNAP History (LSTCNPHST): This command lists the CSNAP history in the system.

System Verification Procedures

The following are the system verification procedures:

- Start Problem Determination Procedure (STRPDP): This command provides the user access to a set of diagnostic procedures that run below the machine interface to determine if CPF can function.
- Verify Printer (VFYPRT): This command is used to produce a print pattern on a 5219/5224/5225/5256 Printer for a specified number of times.

Service Relationship to Other CPF Components

Figure SC-1 shows the relationship between the service component and other CPF components. The following components use service for the indicated purpose:

1 Work control uses service during the start CPF process to allow programming change application or removal. Work control also uses service to terminate and clean-up alert message processing.

2 Message handler uses service to forward an alert message to the network host.

3 Work monitor uses service to clean-up alert message processing in the QLUS process (when terminating). Work monitor also invokes service to handle service request events.

The following components are used by service for the indicated purpose:

4 Service uses spooling to save spooled files for the APAR function.

5 Service uses the command analyzer component to pass command parameters to the service CPPs (command processing programs).

6 Service uses the save/restore component to save and restore objects for the APAR function and the programming change function.

7 Service uses the librarian component to obtain addressability and system object information.

8 Service uses the concurrent service monitor component to provide access to the machine service functions.

9 Service uses the message handler component to signal exceptions, send messages, and to receive message data.

10 Service uses the work control component to gain access to job structure components, such as the work control block and work control block table.

11 Service uses the security component to check the user's authorization to dump objects, to apply, remove, and display programming changes, to patch programs, and to materialize and encapsulate programs.

12 Service uses the common data management component to output dumps, traces, and displays.

13 Service uses console function manager component subfile support to display the deferred programming change screen at start CPF time, to display the programming change and patch status displays, and to display the programming change detailed information display.

Service may also use other components' *special object dump routines* to dump objects such as files. These routines use service to format and output the dumps. Service gets addressability to the special object dump routines through the librarian component.

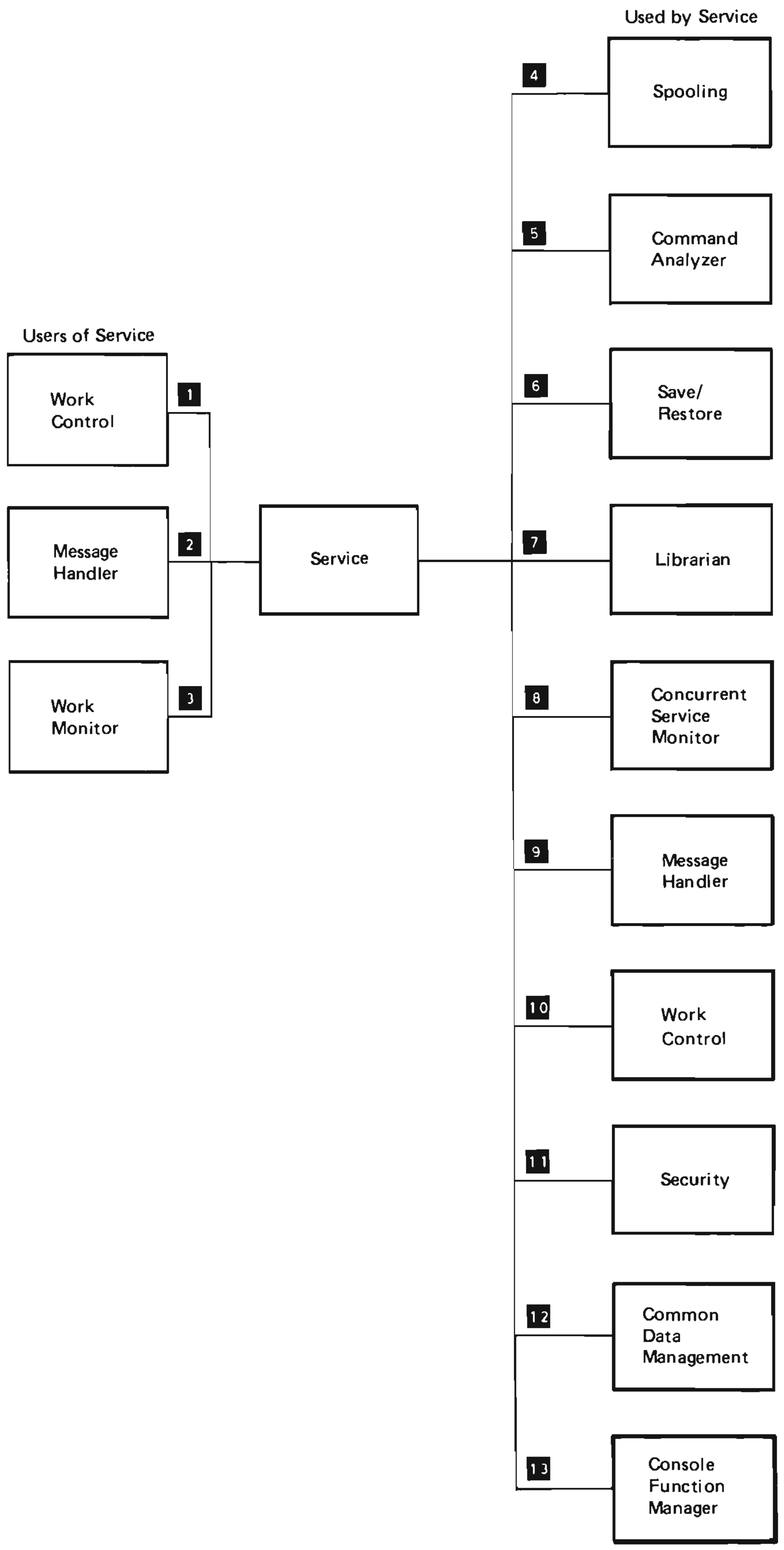


Figure SC-1. Service Relationship to Other CPF Components

Dump Current Job

Figure SC-2 shows an overview of the dump current job operation.

1 The Dump Job (DMPJOB) command is decoded by the command analyzer and control is transferred to QSCDJOB.

2 QSCDJOB obtains addressability to the SCO (service communications object) and determines if the job is in service mode.

3 If the job is *not* in service mode, control is transferred to QSCGJOB.

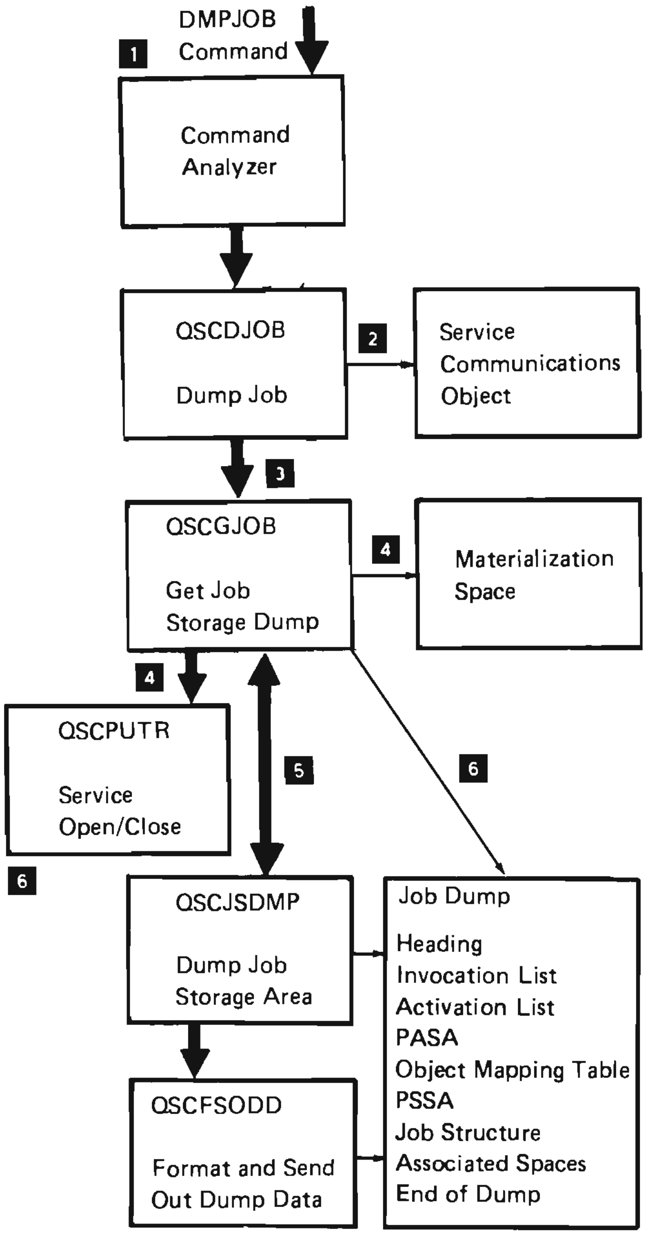


Figure SC-2. Dump Current Job Overview

4 QSCGJOB, using QSCPUTR, opens the printer device file, sends the dump heading information, initializes the default materialization space, and calls QSCJSDMP.

5 QSCJSDMP, using QSCFSODD, sends out the invocation list, activation list, PASA, PSSA, and the job structure objects. Control is returned to QSCGJOB.

6 QSCGJOB sends out the ending dump record, uses QSCPUTR to close the printer device file, destroys the default materialization space, and destroys the SCO.

7 Control is returned to the caller.

Dump Serviced Job

Figure SC-3 shows an overview of a dump serviced job operation.

1 The command analyzer decodes the Dump Job (DMPJOB) command and transfers control to QSCDJOB.

2 QSCDJOB obtains addressability to an SCO (service communications object) and determines if the job is in service mode. If the job *is* in service mode, a service message is sent to the serviced job and is transferred to QSCDH.

3 QSCDH receives service data from the serviced job through the service reply queue. QSCDH tells the serviced job to send more data through the service command queue. The service command queue and the service reply queue are addressable through the SCO. Service data can be in four forms: open file request, data put request, program message request, and close file request. After a closed file request is serviced, control is returned to the CPP caller. An escape program message request results in an escape message being sent to the caller of QSCDJOB.

4 Control is returned to the caller.

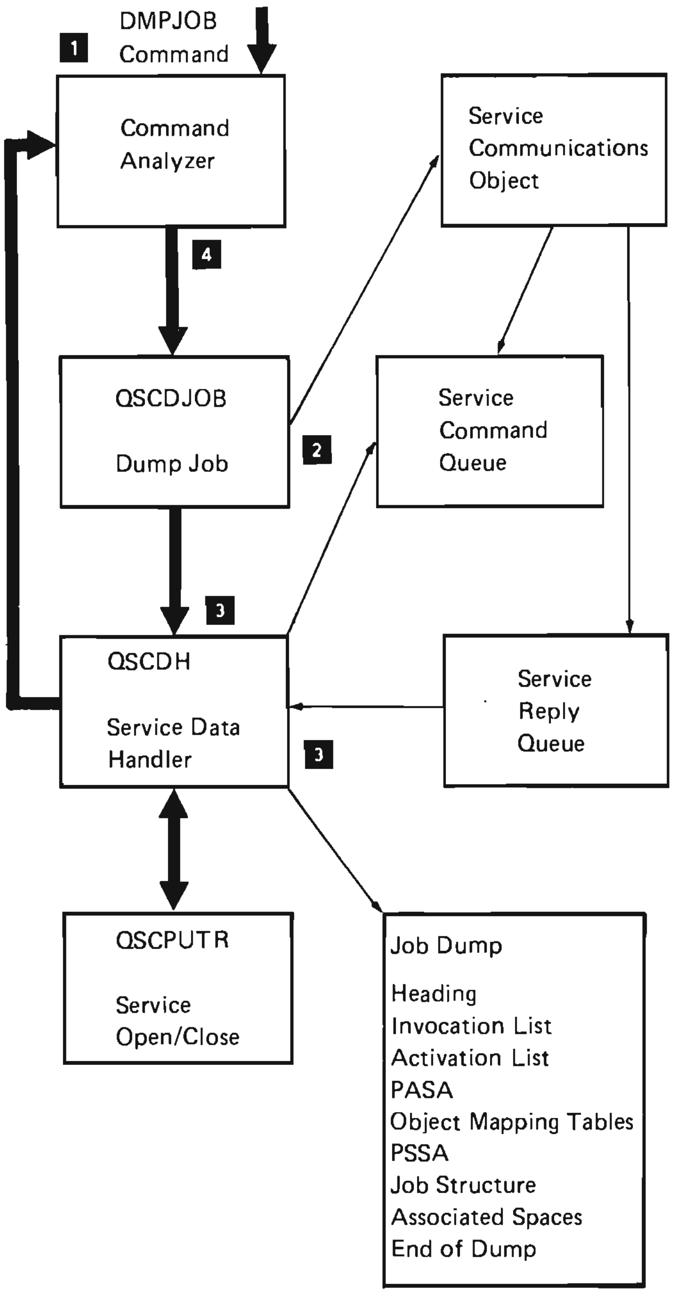


Figure SC-3. Dump Serviced Job Overview

Service Job Command

Figure SC-4 shows an overview of the Service Job (SRVJOB) command operation. The Service Job command is used to put a job in service mode so that other service commands can be entered into the specified job.

1 The Service Job command is decoded by the command analyzer and control is transferred to QSCSVJOB. The job name is passed as an input parameter.

2 The WCBT (work control block table) is searched for the job name. The service index is located and the job name is inserted. An SCO (service communications object) is obtained (by creation or resolution to an existing one) and the service, command, and reply queues are created; the command queue for routing commands to the serviced job, and the reply queue for receiving responses from the serviced job are addressed through the SCO. An event is signaled to the serviced job, passing a pointer to the SCO in that job as event data. The servicer goes into a wait on the reply queue. A response completes the operation.

3 Control is returned to the caller.

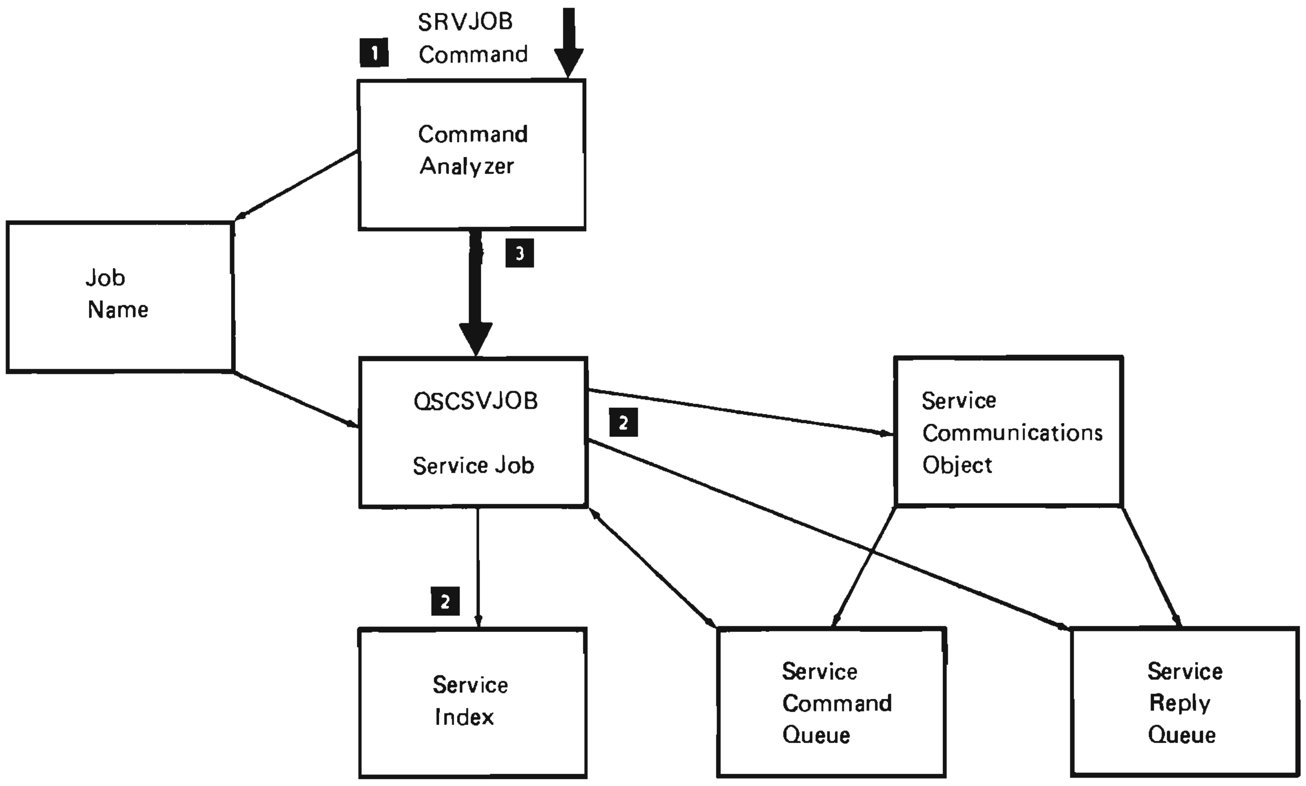


Figure SC-4. Service Job Command Overview

Dump Object or Dump System Object

Figure SC-5 shows an overview of a dump object or dump system object in a current job operation.

1 The command analyzer decodes a Dump Object (DMPOBJ) or Dump System Object (DMPSYSOBJ) command and transfers control to QSCDOBJ if an external object is to be dumped or to QSCDSO if a system object is to be dumped.

2 An SCO (service communications object) is obtained (either created or resolved to) and a pointer to it is put in its parameter area. The command input parameters are also put into the SCO parameter area.

3 Control is transferred to QSCGSO with the pointer to the SCO passed as an argument.

4 QSCGSO gets a specific object or a list of objects, dumps each object, and then control is returned to the caller.

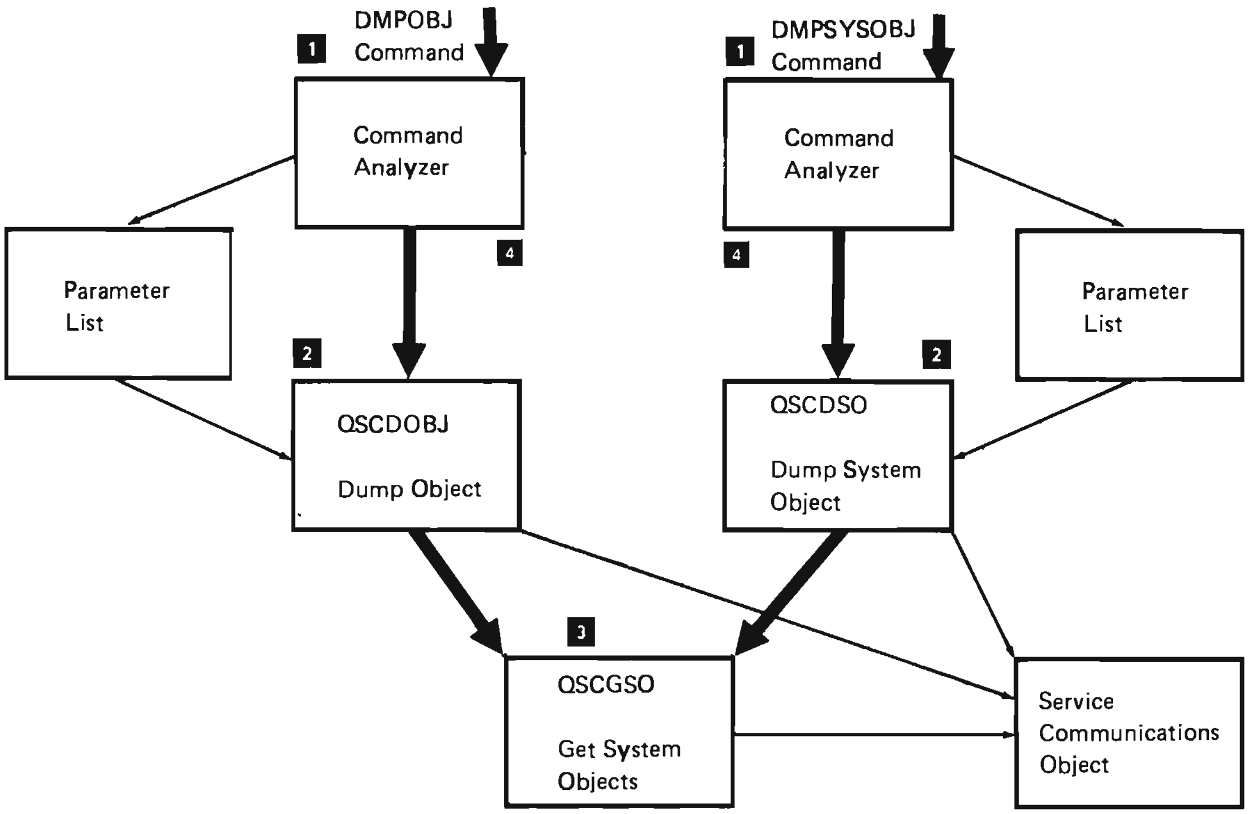


Figure SC-5. Dump Object/System Object Overview

Dump Serviced Job Object or System Object

Figure SC-6 shows an overview of a dump object or system object in a serviced job operation.

1 The command analyzer decodes a Dump Object (DMPOBJ) or Dump System Object (DMPSYSOBJ) command and transfers control to QSCDOBJ if an external object is to be dumped or to QSCDSO if a system object is to be dumped.

2 The SCO (service communications object) that was set up by the QSCSVJOB module is located and because another job is being serviced, a message is sent to the serviced job.

3 QSCDH handles data and messages sent from the serviced job. QSCDH issues a dequeue with wait against the reply queue. The serviced job then puts the information in the associated space of the service reply queue (data queue for the serviced job) and the service message is enqueued to the reply queue by the serviced job to satisfy the dequeue executed by QSCDH. QSCDH, using QSCPUTR to open and close the printer device file, writes the dump data to the printer file. QSCDH returns to the caller of QSCDOBJ or QSCDSO after a close message or an escape program message is processed.

4 Control is returned to the caller.

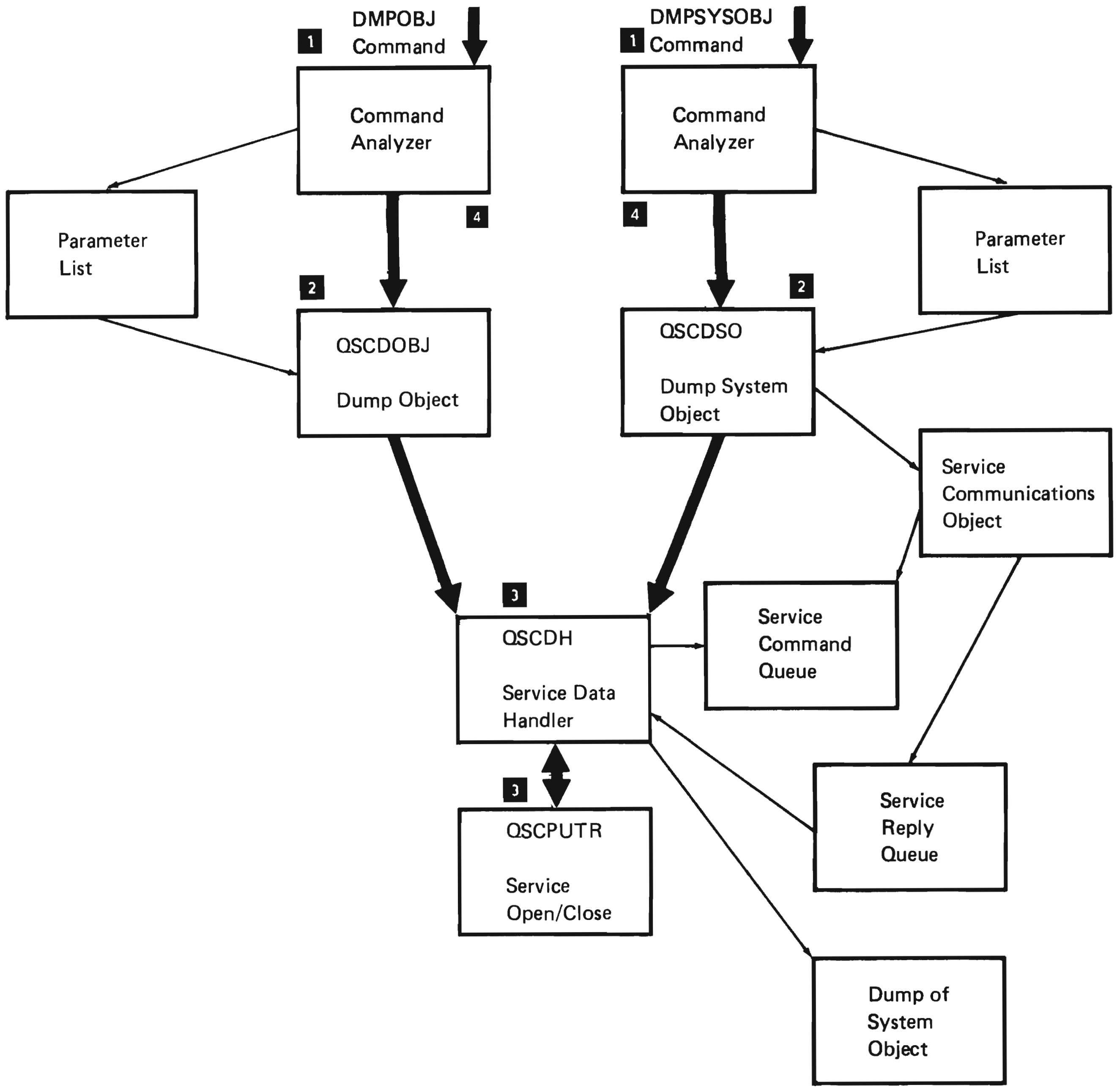


Figure SC-6. Dump Serviced Job Object/System Object Overview

Dump Job Command Executed in a Serviced Job

Figure SC-7 shows an overview of the execution of the Dump Job command in a serviced job.

1 QSCGJOB is called by QSCRH. The SCO (service communications object) is passed as input.

2 A materialization space is set up and addressability to it is put into the SCO. Service messages are put into the associated space of the service data queue. These messages are used to open the printer device file and to print out dump heading lines.

3 QSCJSDMP is called to dump the job storage areas. The dump data is sent to the servicing job through the associated space of the service data queue. When the space is full, a dummy message is enqueued to the service data queue and QSCJSDMP waits for a reply on the service request queue. After processing the data, the servicing job enqueues a response on its command queue (request queue for the serviced job).

4 QSCFSODD is called to format and dump the hexadecimal data using the data queue. Control is then returned to QSCJSDMP.

5 QSCJSDMP returns control to QSCGJOB. End-of-dump lines and a message to close the print device files are sent to the servicing job and the materialization space and SCO are destroyed.

6 Control is returned to the caller.

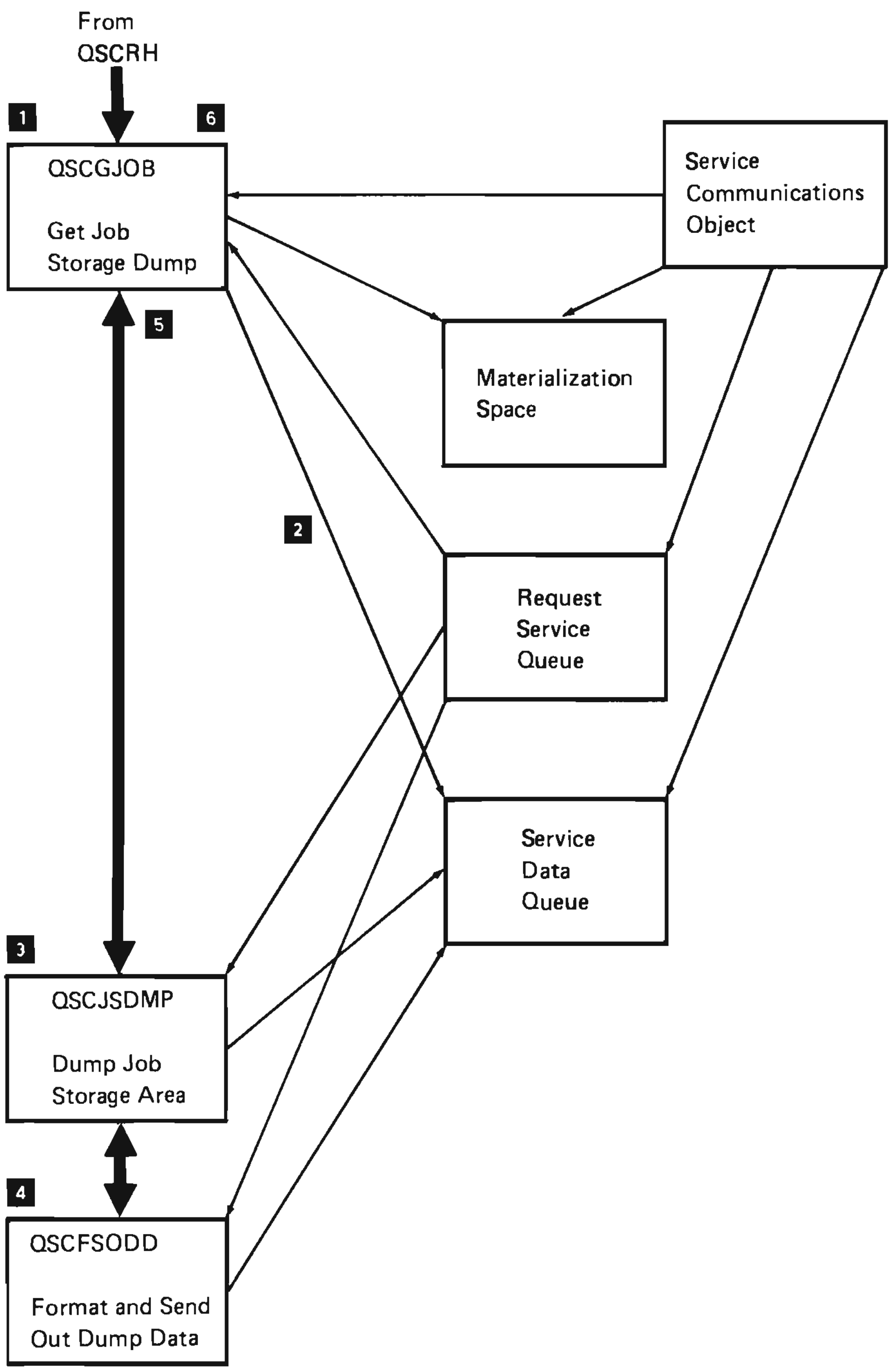


Figure SC-7. Dump Job Execution in a Serviced Job Overview

Get System Object

Figure SC-8 shows an overview of the get system object operation.

1 QSCGSO is called and receives the SCO (service communications object) as input. The command parameters for the dump are in the SCO.

2 A materialization space is created, initialized, and its address put in the SCO. Pointers to the objects to be dumped are obtained by resolving to the object or, for generic dumps, by invoking the librarian component.

3 If the request is not generic and a list of offsets is specified, QSCGSO determines the object to be dumped. The first offset is combined with the addressability of the base object to determine the addressability of the next object. Each succeeding object's addressability is similarly determined until the last offset is used; the last object addressed is dumped.

4 QSCPUTR is called to open the printer device file and QSCGSO provides the dump heading.

5 If the space parameter is specified or the final pointer located in **3** is a space pointer, QSCGSO determines the length of the space to be dumped and QSCSPCDM is called to dump the space.

6 QSCOBJDM is called to dump each object specified other than spaces and the machine context. The pointer to the object is put into the SCO and the materialization space is used for each object.

7 QSCMMCTX is called instead of QSCOBJDM if the machine context is to be dumped.

8 QSCFSODD is called by QSCMMCTX, QSCOBJDM, and QSCSPCDM to format and dump data in hexadecimal and EBCDIC formats.

9 QSCPUTR is called to close the printer device file; the SCO and materialization space are destroyed.

10 If an error is detected during the dump, QSCGSO sets up exception data and signals an exception to QSCGSO's caller.

11 Control is returned to the caller.

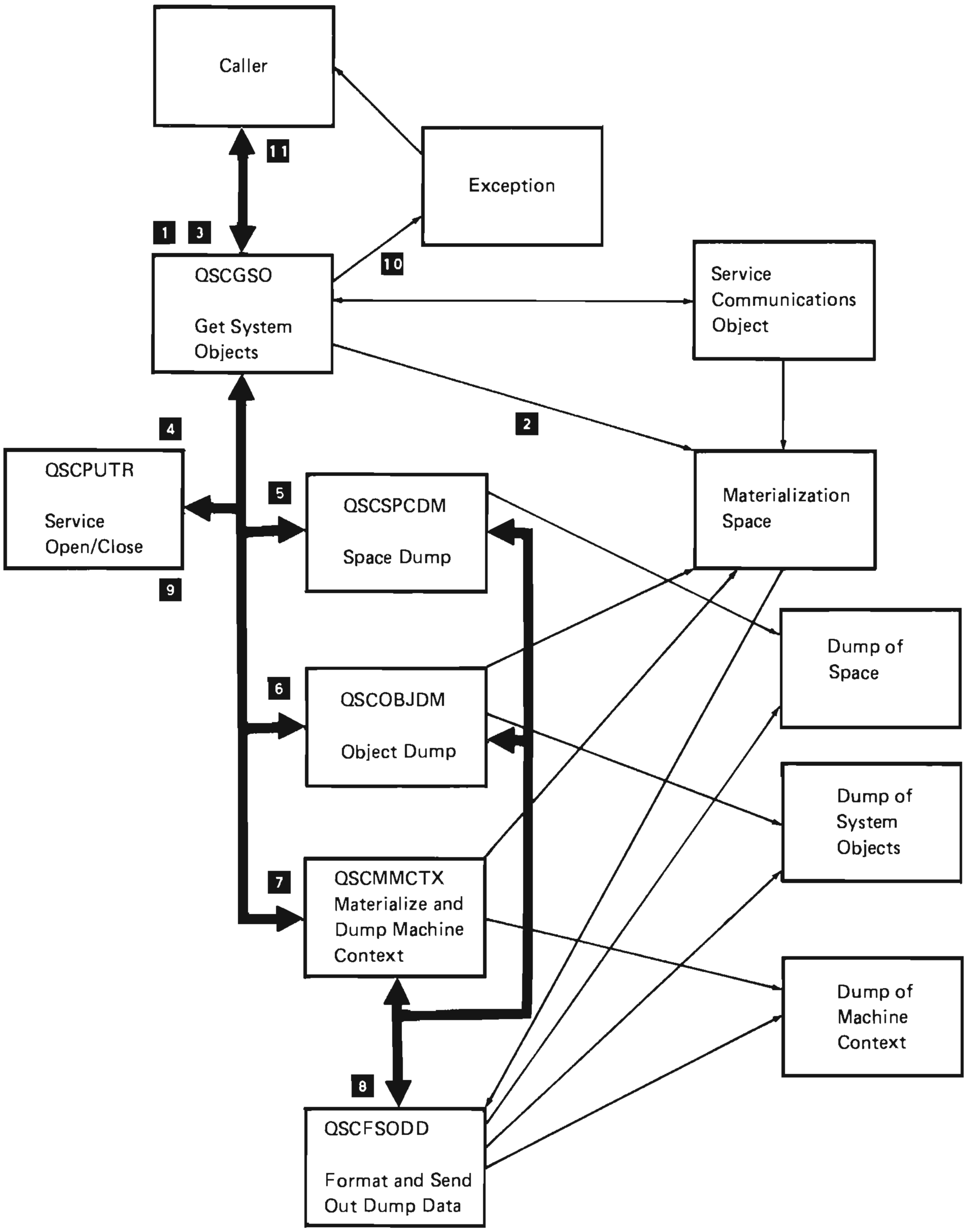


Figure SC-8. Get System Object Overview

Dump Object

Figure SC-9 shows an overview of the object dump operation.

1 QSCOBJDM is called. The SCO (service communications object) is passed as input. The SCO contains a pointer to a common materialization space and a pointer to the object to be dumped.

2 QSCOBJDM checks the user's authority to dump the object and locks the object, if necessary. The object heading line is sent to the printer device file.

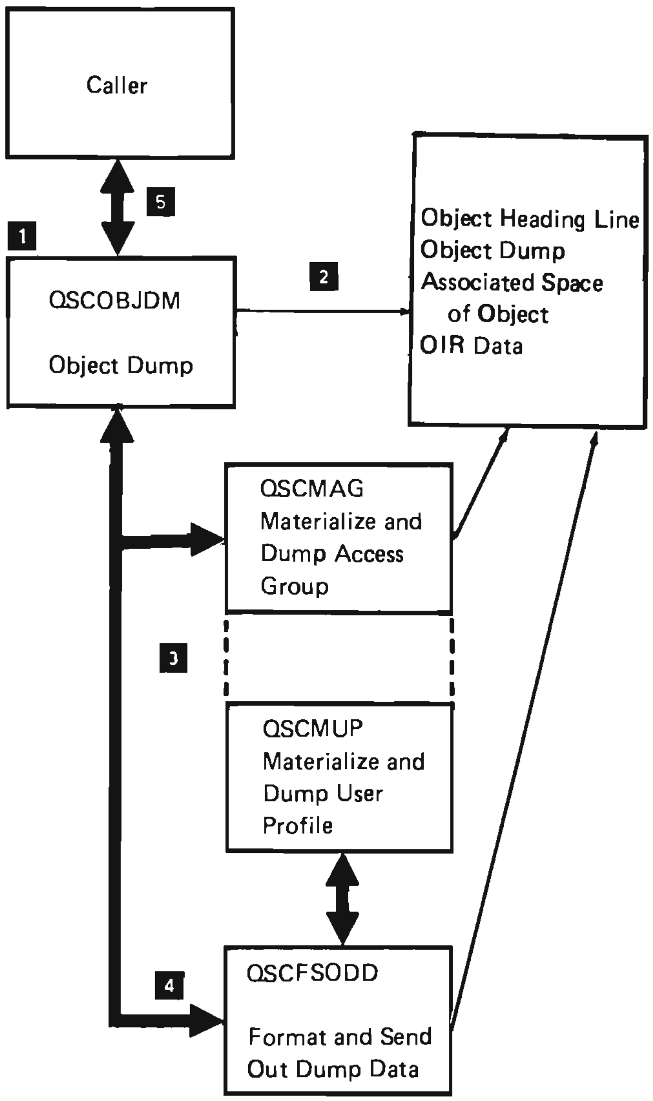
3 The specific dump routine for each machine interface object type is called to materialize and dump that machine interface object. QSCFSODD is called by the dump routines to put out the materialized data. The dump routines are:

- QSCMAG–Materialize and dump access group
- QSCMCB–Materialize and dump control block
- QSCMCD–Materialize and dump controller description
- QSCMCTX–Materialize and dump context
- QSCMCUR–Materialize and dump cursor
- QSCMDMPS–Materialize and dump dump space
- QSCMDS–Materialize and dump data space
- QSCMDSI–Materialize and dump data space index
- QSCMIDX–Materialize and dump independent index
- QSCMJPRT–Materialize and dump journal port
- QSCMJSPC–Materialize and dump journal space
- QSCMLUD–Materialize and dump logical unit description
- QSCMND–Materialize and dump network description
- QSCMPGM–Materialize and dump program
- QSCMQ–Materialize and dump machine interface queue attributes
- QSCMSP–Materialize and dump space attributes
- QSCMUP–Materialize and dump user profile

Note: If the object to be dumped is a composite object, QSCOBJDM calls itself to dump each component object of the composite. If the object is not standard, the designated dump routine for the object is called.

4 If present, the OIR (object information repository) data is dumped. QSCFSODD is called to dump the byte space of space objects and the associated space of system objects.

5 Control is returned to the caller.



Caller
5
1
QSCOBJDM
Object Dump
2
Object Heading Line
Object Dump
Associated Space of Object
OIR Data
QSCMAG
Materialize and Dump Access Group
3
QSCMUP
Materialize and Dump User Profile
4
QSCFSODD
Format and Send Out Dump Data

Figure SC-9. Dump Object Overview

Service Alert Event Handler

Figure SC-10 shows an overview of service alert event handler.

1 Message handler signals an alert event to the QLUS if alert processing is active and the message is to be alerted.

2 QSCAEVTH builds a temporary space for the alert event.

3 QSCAEVTH issues a REQIO instruction to have the vertical microcode send the alert message to the network host.

4 QSCAEVTH dequeues the REQIO feedback record from the machine interface response queue of the QLUS process, and returns control.

5 Work control signals an alert termination event if alert processing is terminated via the Change Network Attributes command. This allows QSCAEVTH to destroy the REQIO spaces and to remove from the queue any feedback records for alert REQIOs from the machine interface response queue. Any alert processing REQIOs that have been sent to the vertical microcode but have not been finished are canceled.

6 Work monitor calls QSCAEVTH if alert processing is active and the QLUS process is terminating. QSCAEVTH then performs clean-up as in 5.

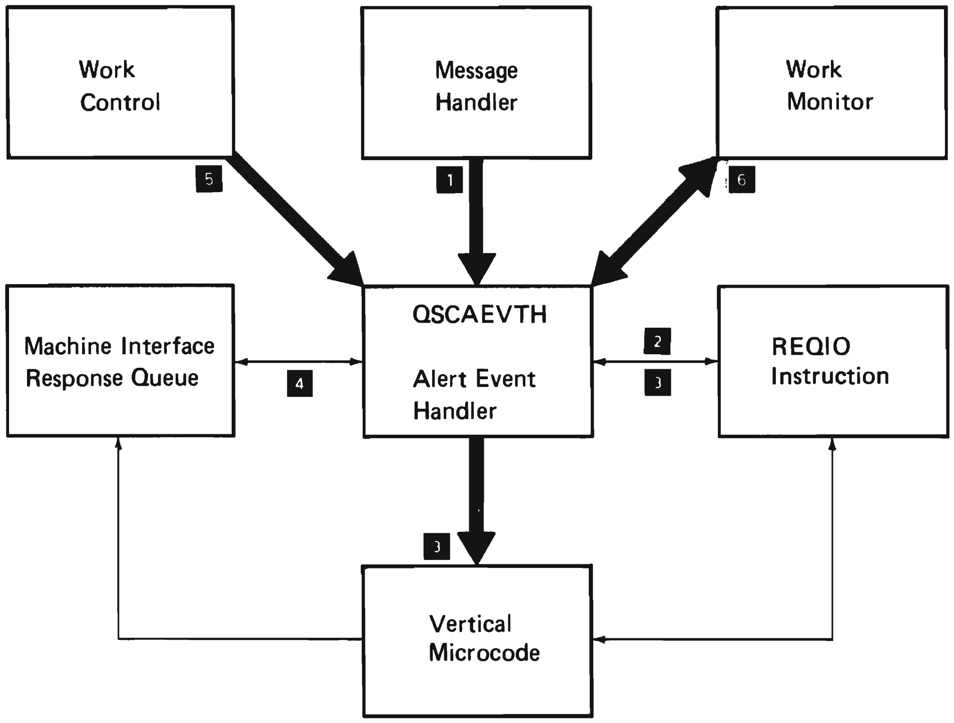


Figure SC-10. Service Alert Event Handler Overview

Service Request Event Handler

Figure SC-11 shows an overview of the service request event handler.

1 A service event is signaled to the serviced job by CPPs (command processing programs) executing in the servicing job. The event data is passed to QSCRH in a service message.

2 Work monitor's process control event handler invokes QSCRH to process the service request event. QSCRH, depending on the event data sent by the signaler, performs one of the following:

3 Using a pointer to the SCO of the servicing job, addressability to the service queues is obtained and set in the new SCO. The service command queue becomes the service request queue and the service reply queue becomes the service data queue in the serviced job. The address of the new SCO is put into the SCO of the servicer and a completion message is sent using the service data queue.

4 The SCO is destroyed.

5 QSCTJOBR is called to activate the CPF call/return trace.

6 QSCGJOB is called to execute the dump job functions.

7 QSCGSO is called to execute the dump system object functions.

8 QSCGJI is called to execute the dump job internal function.

9 QSCWMINT is called to perform process initiation time service functions, including activation of the CPF call/return trace using QSCTJOBR.

10 In addition, if the serviced process terminates, QSCRH is called in the service process to terminate service mode and send a process terminated message to the user.

11 Control is returned to the caller.

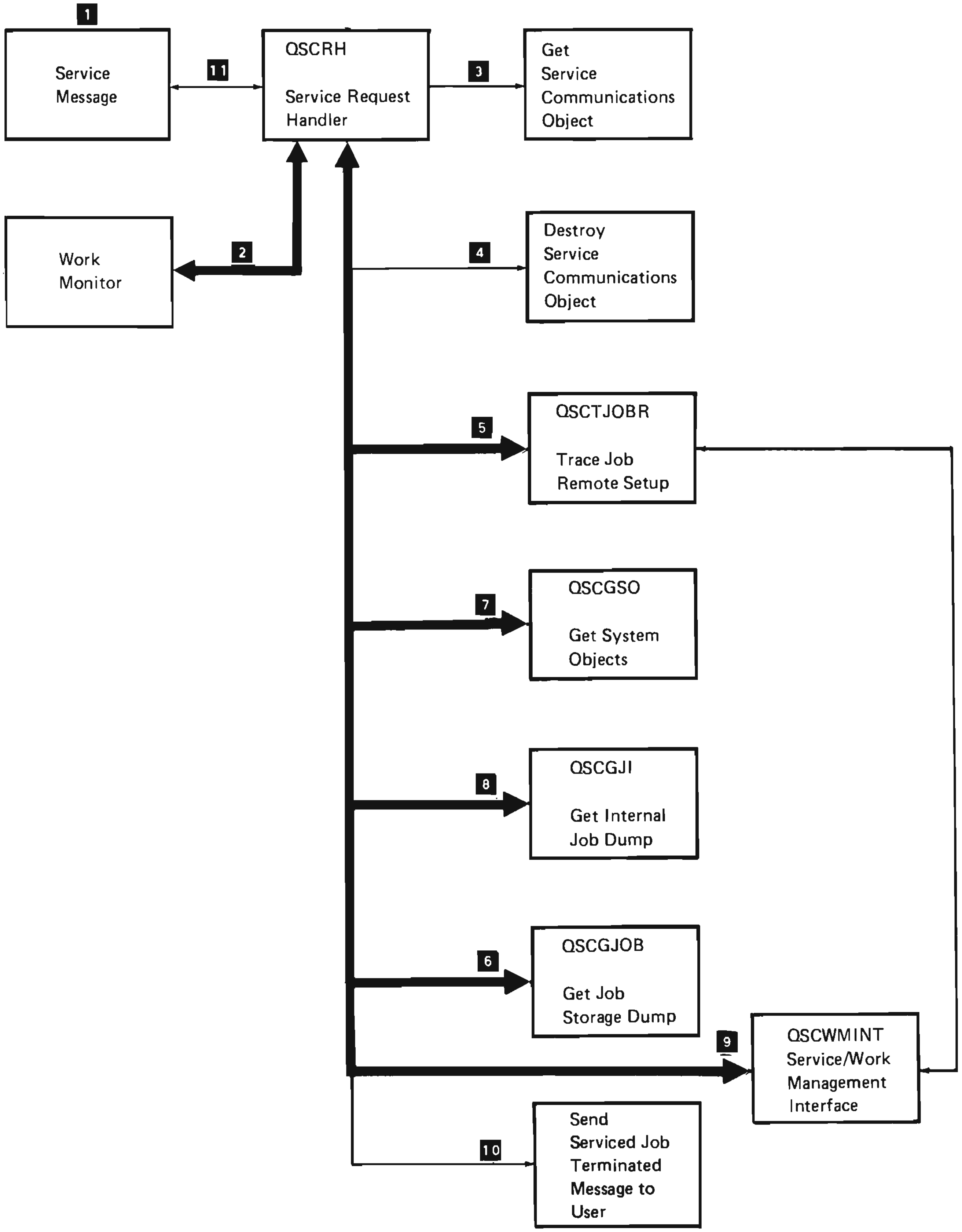


Figure SC-11. Service Request Event Handler Overview

Service Data Handler

Figure SC-12 shows an overview of the service data handler functions.

1 QSCDH is transferred by a dump CPP.

2 A dequeue with wait from the service reply queue is executed. When the dequeue is satisfied, data put in the associated space of the reply queue is processed as follows:

3 If the message type is open or close, QSCPUTR is called to open or close the printer device file.

4 If the message type is put, the data is sent to the printer to be dumped. If the end of the associated space is reached, the offset to the next message is reset to the start of the associated space, a message is enqueued to the command queue and step **2** is reexecuted. The serviced job, after filling up the associated space, sends its data queue message, and waits on a reply at its request queue.

5 If the message type is send program message, then the specified program message is sent to the program message queue of the program which called CPP and QSCDH.

6 If the message type is close or send program message and the program message type is escape or completion, then control is returned to the caller.

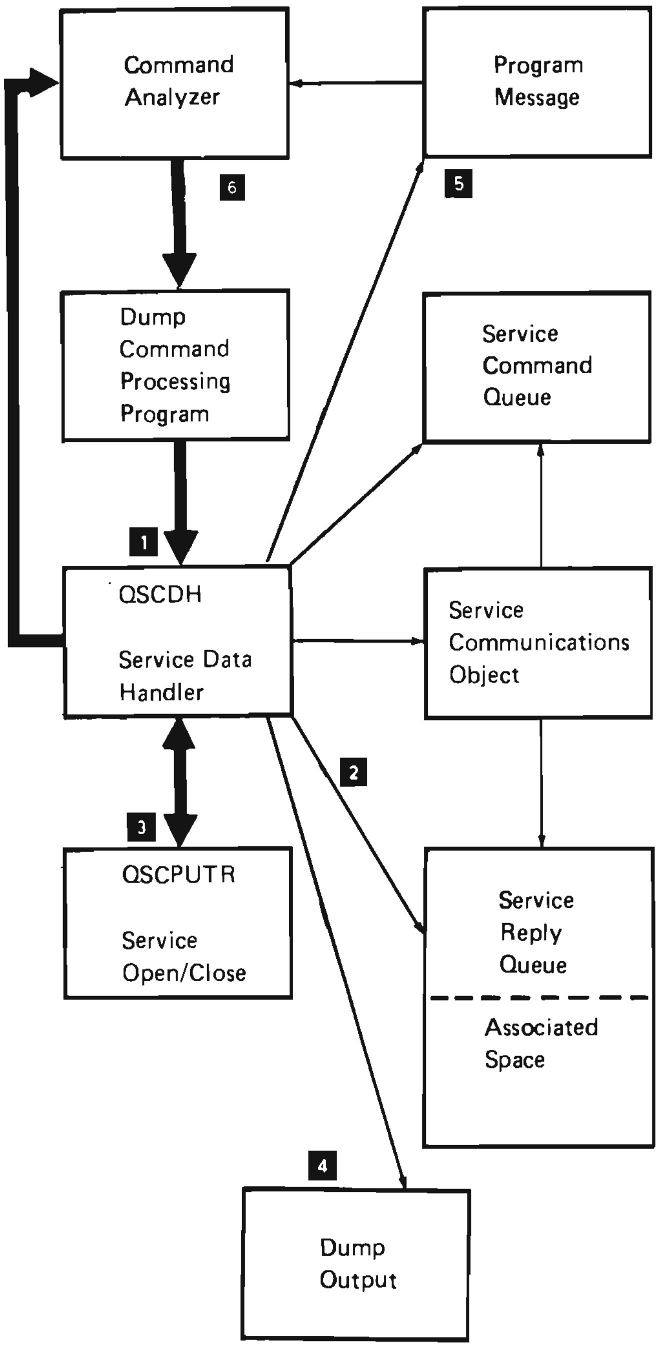


Figure SC-12. Service Data Handler Overview

Trace Job Command

Figure SC-13 shows an overview of the Trace Job (TRCJOB) command operation.

1 The command analyzer decodes the Trace Job command and control is transferred to QSCTJOB passing the command parameters as input.

2 An SCO (service communications object) is obtained and, depending on whether trace is to be set on, set off, or canceled, the following happens:

- If trace is to be set *on*:
  - A trace table is created, dependent on the storage size parameter.
  - If a trace exit program is specified, locate it.
  - Set trace control flags in the SCQ and work control block.
  - If *not* servicing another process, turn on the invocation reference trace.
  - If servicing another process, send a service message to the process that is being serviced.

- If trace is to be set *off*:
  - The trace environment is terminated in the related processes.
  - Trace records are retrieved from the trace table, formatted and sent to a spooled file or a data base output file, or both, and then the trace table is destroyed.
  - If servicing another process, a message is sent to that process. That process can then perform any necessary functions.

- If trace is to be *canceled:*
  - The trace environment is terminated in the related process.
  - The trace table is destroyed (no output records).
  - If servicing another process, a message is sent to that process, which can then perform any necessary functions.

3 Control is returned to the caller.

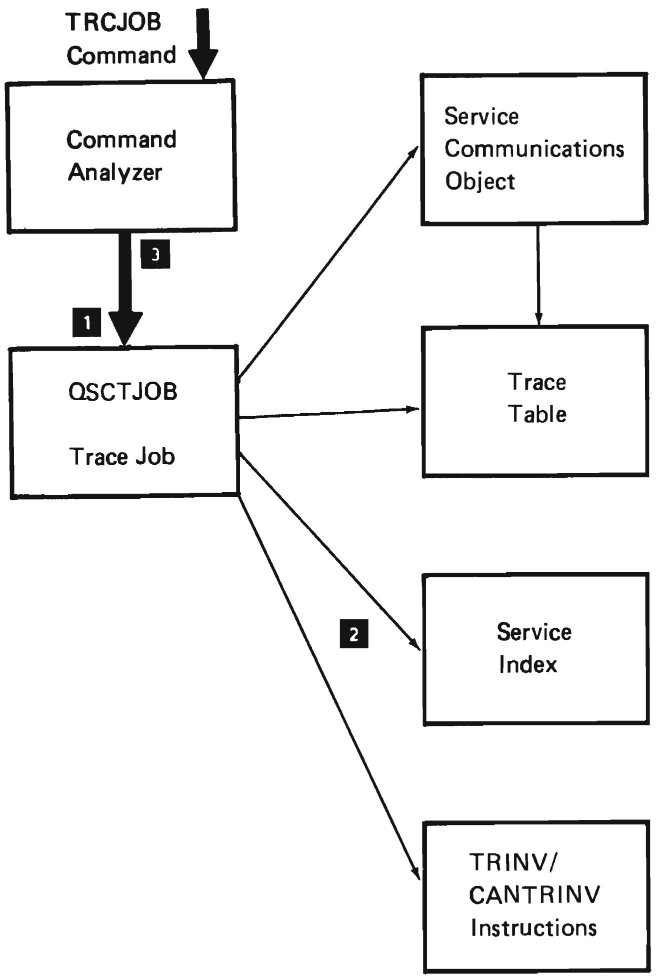


Figure SC-13. Trace Job Command Overview

Trace Job Command in a Serviced Job

Figure SC-14 shows an overview of a Trace Job (TRCJOB) command operation in a serviced job.

1 QSCRH calls QSCTJOBR passing a pointer to a SCO as input.

2 Depending on whether the flag in the SCO has been set on or off by the CPP in the servicing job, one of the following happens:

- If the flag is set *on*:
  - The Trace Invocations instruction is executed for each invocation in the invocation stack.
- If the flag is set *off*:
  - The Trace Invocations instruction is canceled for each invocation in the invocation stack.

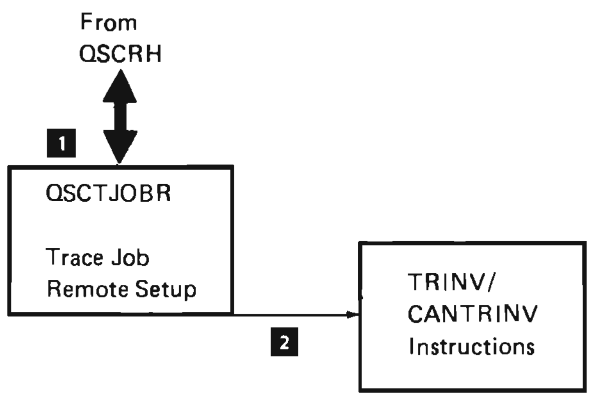


Figure SC-14. Trace Job Command in a Serviced Job Overview

Trace Internal Command

Figure SC-15 shows an overview of the Trace Internal (TRCINT) command operation.

1 The command analyzer decodes the Trace Internal command and control is transferred to QSCTI passing command parameters as input.

2 A command string is built using the input parameters. The internal macro ?CALLS is invoked to establish a link to the concurrent service monitor.

3 The machine internal trace is executed under the concurrent service monitor using REQIOs to the machine service component.

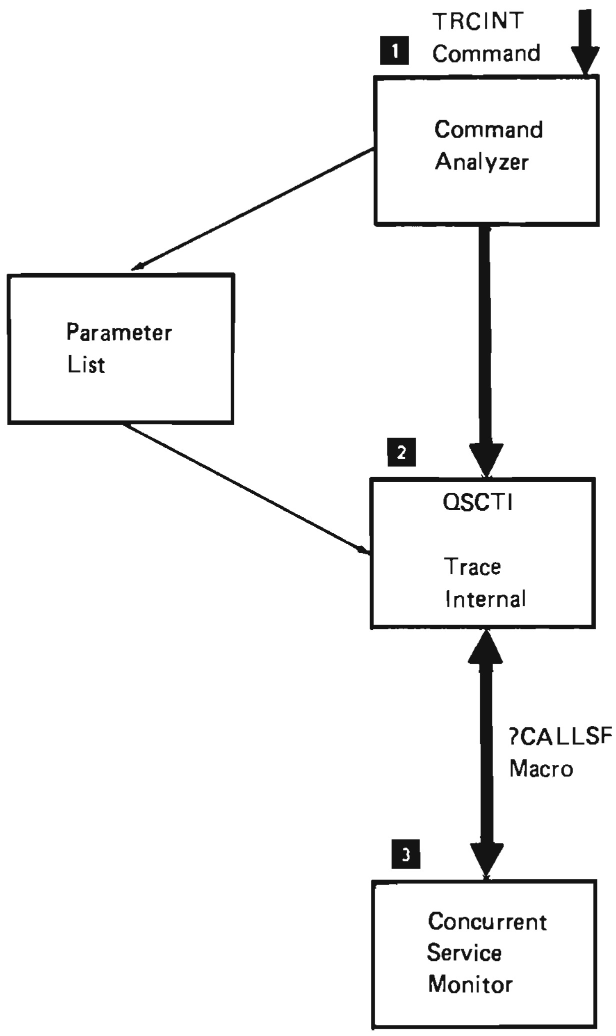


Figure SC-15. Trace Internal Command Overview

This page is intentionally left blank.

Prepare APAR Command

Figure SC-16 shows an overview of the Prepare APAR (PRPAPAR) command operation.

1 The command analyzer decodes the Prepare APAR command and control is transferred to QSCAPAR passing the command parameters as input.

2 If spooled file members are to be included, QSCCOPY is invoked to create physical data base file copies for each member.

3 Save/restore is invoked to save all files from unique libraries first and then to save all programs from unique libraries. Spooled files are saved as separate data base files; the objects are saved by the save/restore component.

4 Any data base files and spaces that were created are destroyed.

5 Control is returned to the caller.

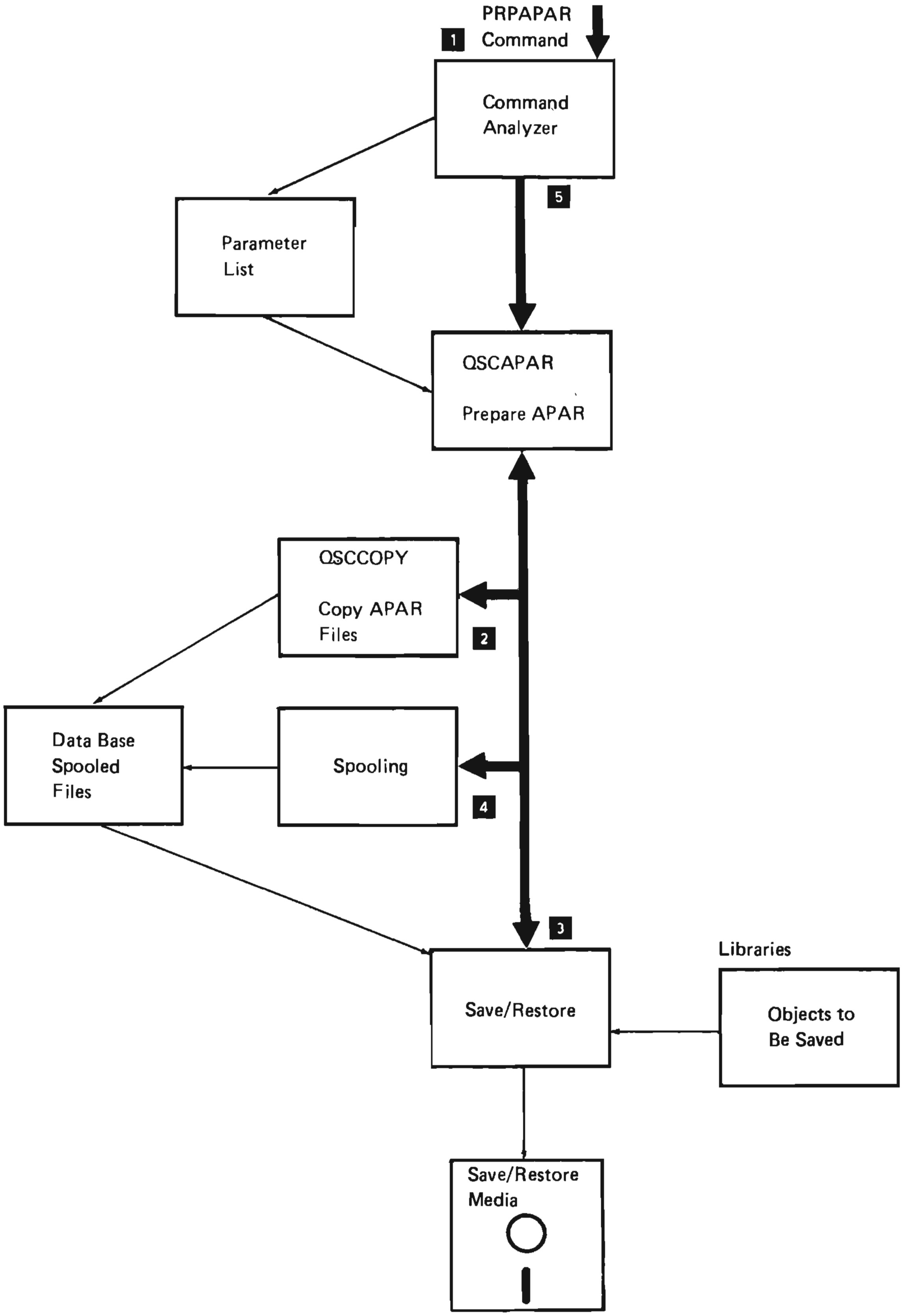


Figure SC-16. Prepare APAR Command Overview

Display Service Status Command

Figure SC-17 shows an overview of the Display Service Status (DSPSRVSTS) command operation.

1 The command analyzer decodes the Display Service Status command and control is transferred to QSCDSS passing the job name as input.

2 The WCBT (work control block table) is searched for the named job. If the name is not found or more than one job has that name, an exception is signaled. A uniquely named SCO (service communications object) is located in the QSYS library. If it is not located, an exception is signaled.

3 The display device (console or work station display) is opened and the job name is placed in the output buffer. Using the WCBTE (work control block table entry) offsets for the servicing and serviced jobs contained in the SCO, the job names are extracted from the WCBTE and also put into the output buffer. If trace is active, the trace table, addressed through the SCO, is accessed for trace information. That trace information is also put into the output buffer.

4 Service status is then displayed to the user.

5 Control is returned to the caller.

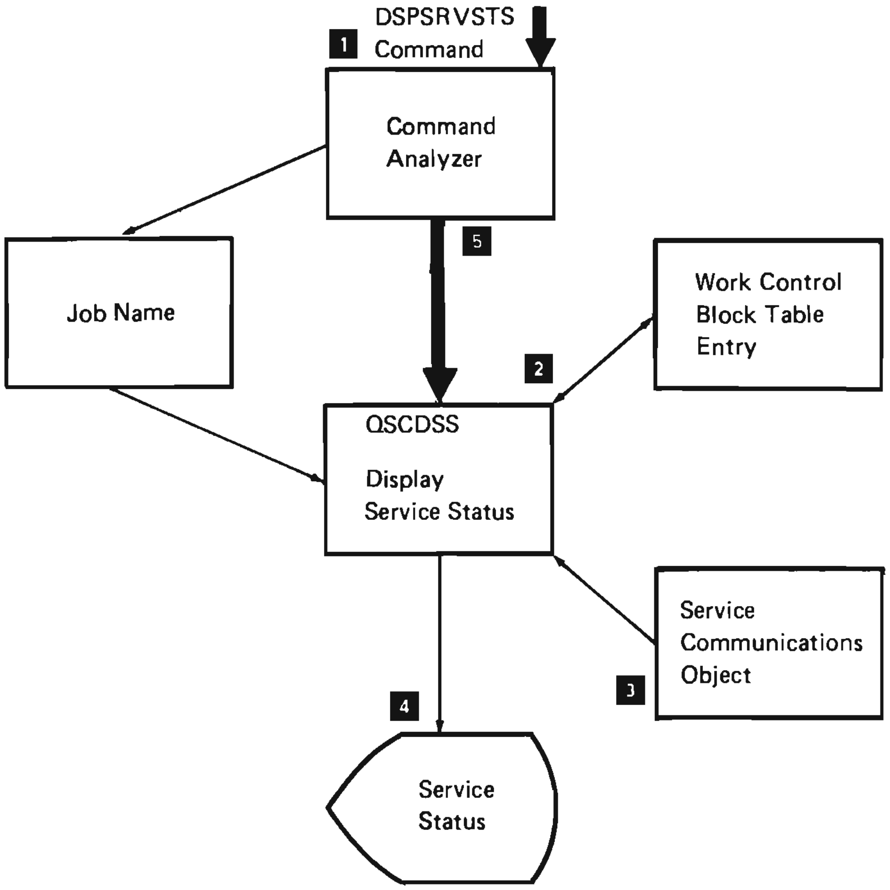


Figure SC-17. Display Service Status Command Overview

This page is intentionally left blank.

Patch Program Command

Figure SC-18 shows an overview of the Patch Program (PCHPGM) command operation.

1 The command analyzer decodes the Patch Program command and control is transferred to QSCPP. The command parameters are passed as input.

2 Using the program name parameter, the program to be patched is addressed and materialized into a created space.

3 The materialized program template is patched as follows:

A The patch data (offset, old data, new data, and check data) may be specified as command parameter input.

B The QPCHSRC source file may be specified as the source of the patch data.

C If neither patch data nor a source file is specified and the job is an interactive job, portions of the program template are displayed to the user. The patch data can then be entered on the display. This also lets the user randomly scroll and patch the program template. Command function keys are provided to end or cancel the patch session.

D A log record is built for each patch and then enqueued to a temporary machine interface queue.

For both command and source file input, old data entered is compared with old data currently at the specified offset. The new data will overlay the old data at the specified offsets. Each offset specifies a separate patch and each row of the interactive display specifies a separate patch.

E A new program is created (encapsulated) using the patched program template. This occurs for command input when all offsets are processed, for source file input when all input records are processed, and for interactive input when the CF3 key is pressed. The new program is inserted into the programming change index for testing by QSCARPC.

F The materialization space is destroyed, the patch log records are dequeued, logged to the change log and printed, and then the queue is destroyed.

G Control is returned to the caller.

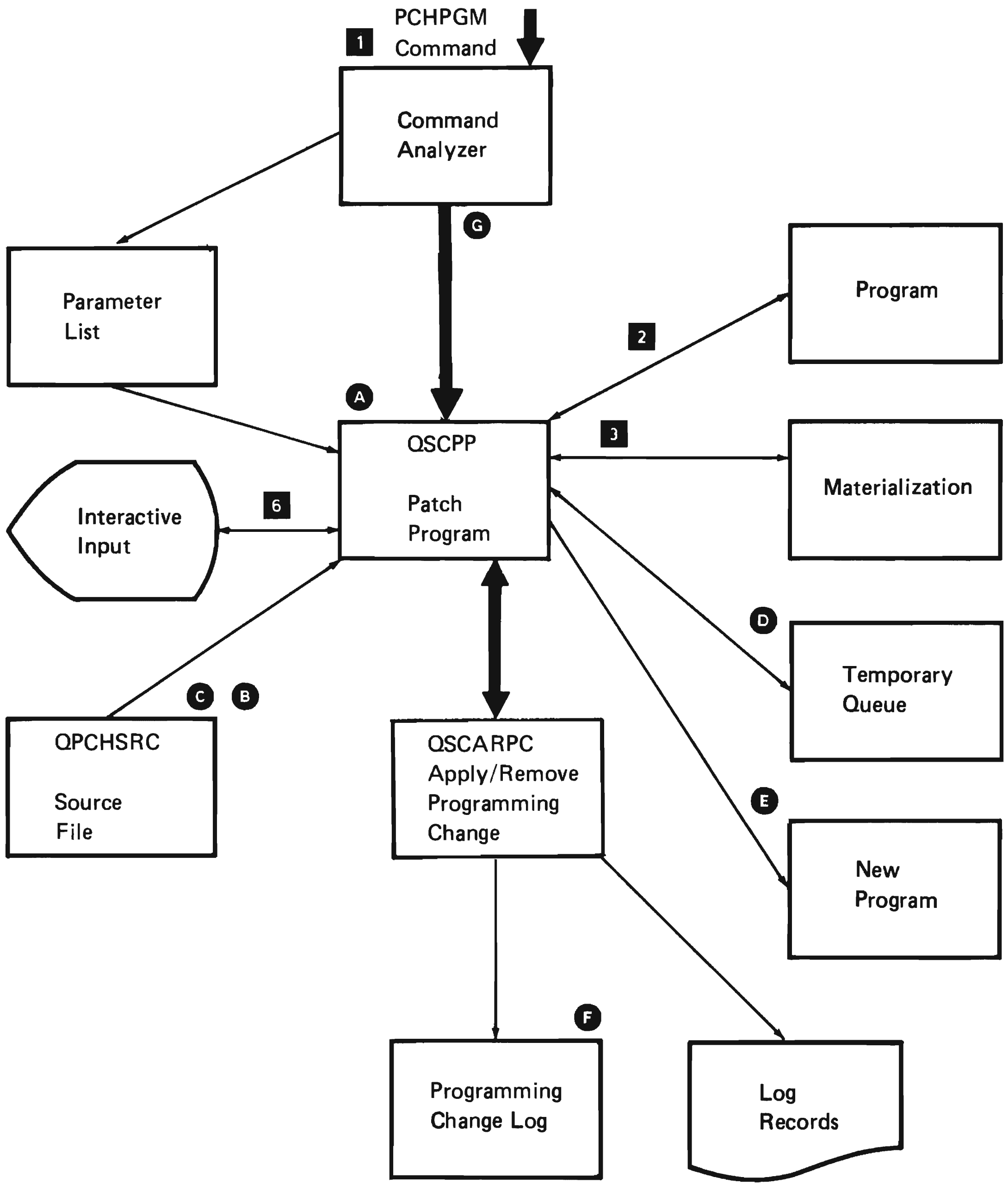


Figure SC-18. Patch Program Command Overview

Patch Program Command Support

Figure SC-19 shows an overview of the support used by the Patch Program command.

1 QSCPP invokes the macro ?LOCPGM. The command parameters are passed as input.

2 ?LOCPGM calls QSCARPC to locate the program object as specified by the Patch Program command. The address of the program object is resolved.

3 QSCARPC returns to QSCPP with the resolved address. If QSCARPC is unable to locate the specified program, a null system pointer is returned.

4 QSCPP invokes the macro ?INSPGM to replace the patched program.

5 ?INSPGM calls QSCARPC to insert the patched program into the system. This object is now a local programming change.

6 Control is returned to the caller.

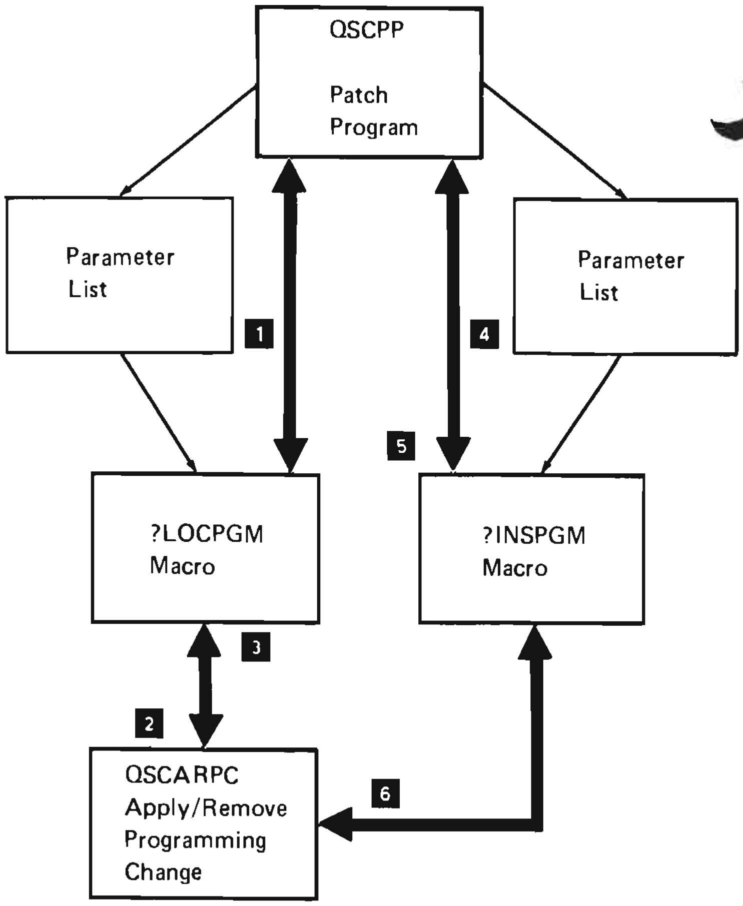


Figure SC-19. Patch Program Command Support Overview

This page is intentionally left blank.

Apply, Remove, and Load Programming Changes

Figure SC-20 shows an overview of the Apply Program Change (APYPGMCHG) command, Remove Program Change (RMVPGMCHG) command, and Load Program Change (LODPGMCHG) command, as well as an overview of applying programming changes at start CPF time.

Apply Program Change

1 The command analyzer decodes an Apply Program Change command and control is transferred to QSCAPC to execute the command.

A QSCAPC calls QSCAUTCK, which verifies user authorization to make programming changes in the specified library. Control is then returned to QSCAPC.

2 QSCAPC checks if the programming change prerequisites have been met. If the prerequisite check fails, an exception message is sent.

3 If the prerequisite check is successful, QSCARPC is called to perform the application of the programming change. If any programming change exit programs exist, they are called. Control is returned to QSCAPC upon successful completion of the change; QSCAPC in turn returns control to the caller.

Remove Program Change

4 The command analyzer decodes a Remove Program Change command and control is transferred to QSCRPC to execute the command.

B QSCRPC calls QSCAUTCK, which verifies user authorization to make programming changes in the specified library. Control is then returned to QSCRPC.

5 QSCRPC determines if the programming change to be removed is a prerequisite that is required to support another programming change. If it is, an exception message is sent.

6 If the prerequisite check is successful, QSCARPC is called to remove the programming change. If any programming change exit programs exist, they are called. Upon successful completion of the removal, control is returned to QSCRPC, which in turn returns control to its caller.

Load Program Change

7 The command analyzer decodes a Load Program Change command and control is transferred to QSCLCPR to execute the command.

C QSCLCPR verifies user authorization to load programming changes into the specified library. Control is then transferred to QSCLPC.

8 QSCLPC determines if the prerequisites for the programming changes about to be loaded are already loaded. If they are not already loaded, an exception message is sent.

9 If the prerequisite check is successful, the program change objects are loaded from the save/restore device. Upon completion of a successful load, (QSCARPC is called to perform the internal load) control is returned to QSCLPC, which in turn returns control to its caller.

Apply or Remove Programming Changes Through the Start CPF Interface

10 The start CPF procedure of work control, using the ?TSTSRV macro expansion, calls QSCCPFI.

11 QSCCPFI determines if there are any deferred programming changes available for the user to perform action upon. These programming changes are displayed to the user. QSCCPFI then monitors the console to determine if the user requests that any of the programming changes be applied. The programming changes that are requested are validity checked. If there are any errors, a message is displayed to the user. All changes must be valid before any processing can take place.

12 When all programming changes are valid, QSCCPFI calls QSCARPC to perform the changes. If any programming change exit programs exist, they are called. Upon successful completion of the changes, control is returned to QSCCPFI, which in turn returns control to its caller.

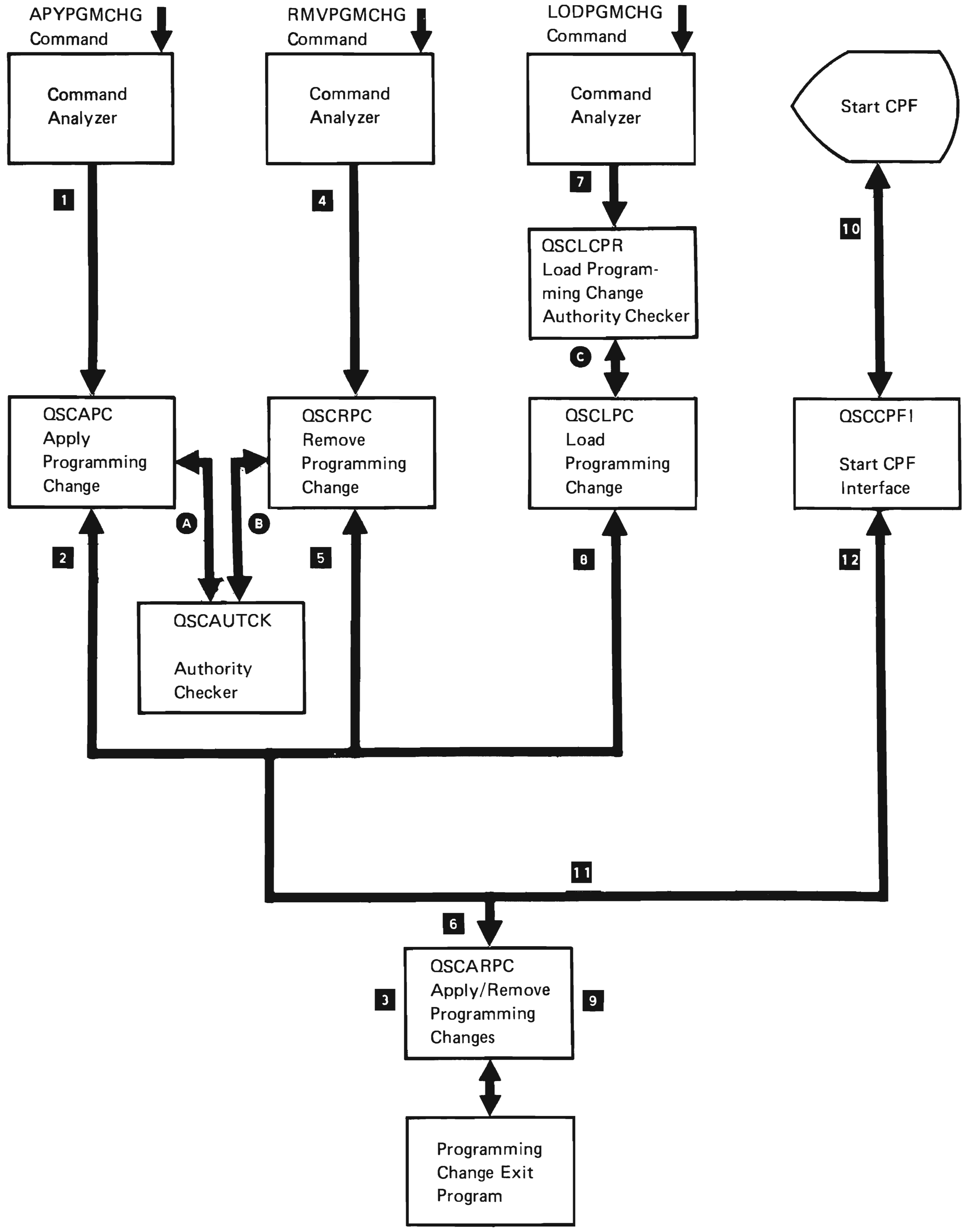

APYPGMCHG Command
RMVPGMCHG Command
LODPGMCHG Command
Command Analyzer
Command Analyzer
Command Analyzer
Start CPF
1
4
7
10
QSCLCPR Load Programming Change Authority Checker
C
QSCAPC Apply Programming Change
QSCRPC Remove Programming Change
QSCLPC Load Programming Change
QSCCPFI Start CPF Interface
A
B
2
5
8
12
QSCAUTCK Authority Checker
11
6
QSCARPC Apply/Remove Programming Changes
3
9
Programming Change Exit Program

Figure SC-20. Apply, Remove, and Load Commands Overview

Display Programming Change Command

Figure SC-21 shows an overview of the Display Programming Change (DSPPGMCHG) command operation.

1 The command analyzer decodes the Display Program Change command and control is transferred to QSCPCSYS. The command parameters are passed as input.

2 QSCPCSYS gathers the status and attributes for all programming change and patches, requested on the command parameters, using the master programming change indexes.

3 QSCPCSYS then produces a status display or a printer listing for the requested programming changes and patches.

4 QSCDSPPC is called if the user requests information regarding programming changes on the status display, or on the command parameters.

5 QSCDSPPC produces a detailed information display or a printer listing for the programming change. When all detailed information displays have been displayed, QSCDSPPC returns control to QSCPCSYS.

6 QSCPCSYS then returns control to the caller.

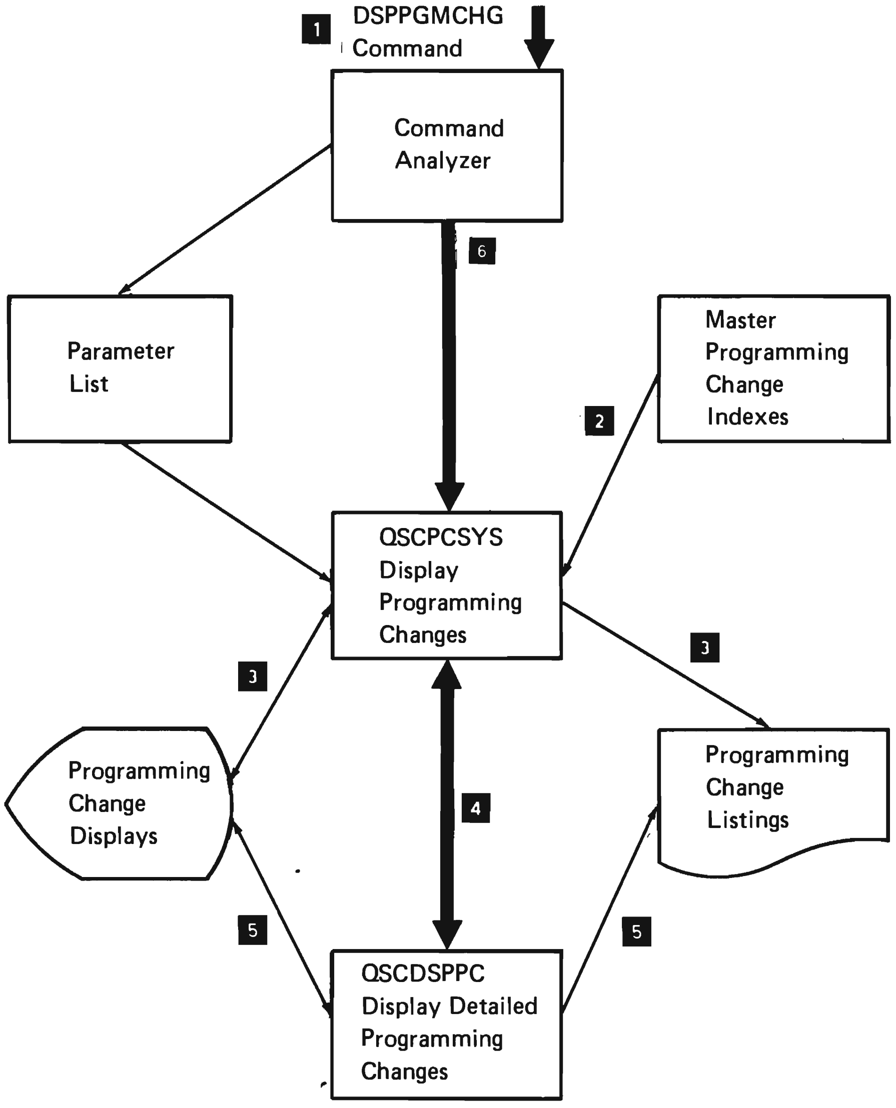

1
DSPPGMCHG Command
Command Analyzer
6
Parameter List
Master Programming Change Indexes
2
QSCPCSYS Display Programming Changes
3
3
Programming Change Displays
4
Programming Change Listings
5
5
QSCDSPPC Display Detailed Programming Changes

Figure SC-21. Display Programming Change Command Overview

Dump Current Job Internal Command

Figure SC-22 shows an overview of the Dump Current Job Internal (DMPJOBINT) command operation.

1 The Dump Current Job Internal command is decoded by the command analyzer and control is transferred to QSCDJI.

2 QSCDJI obtains addressability to the SCO (service communications object) and determines if the job is in service mode.

3 If the job is *not* in service mode, control is transferred to QSCGJI.

4 QSCGJI executes a diagnose instruction to cause an internal process dump to be taken. The dump is written to the internal vertical microcode log by the diagnose instruction.

5 The diagnostic instruction returns the dump ID to QSCGJI.

6 QSCGJI sends the dump ID in the completion message to the requester, and destroys the SCO.

7 Control is returned to the caller.

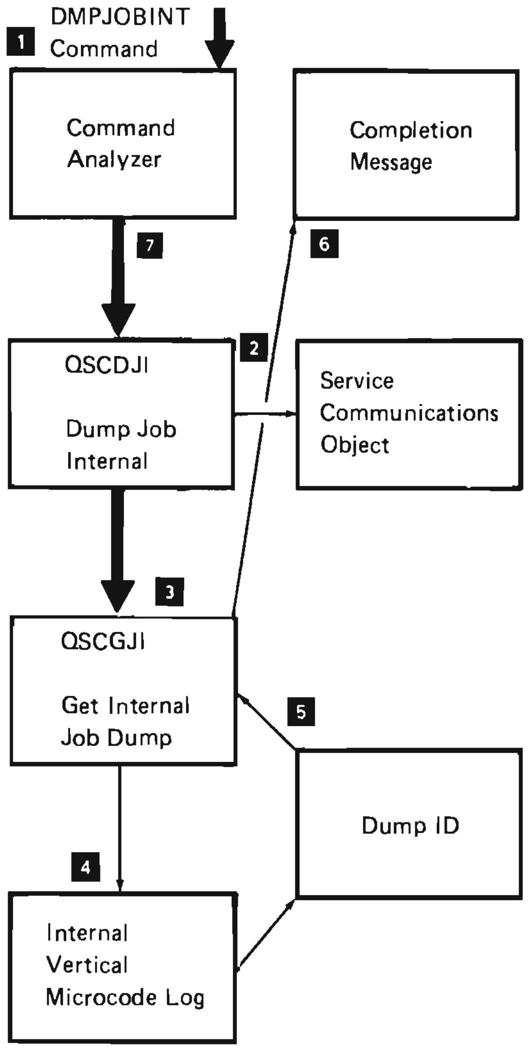


Figure SC-22. Dump Current Job Internal Command Overview

Dump Serviced Job Internal Command

Figure SC-23 shows an overview of the Dump Serviced Job Internal (DMPJOBINT) command.

1. The command analyzer decodes the Dump Serviced Job Internal command and transfers control to QSCDJI.

2. QSCDJI obtains addressability to the SCO (service communications object) and determines if the job is in service mode.

3. If the job *is* in service mode, a service message is sent to the serviced job and QSCDH is transferred.

4. QSCDH receives the completion message containing the dump identifier from the serviced job through the service reply queue.

5. QSCDH sends the completion message to the requester and returns control to QSCDJI.

6. Control is returned to the caller.

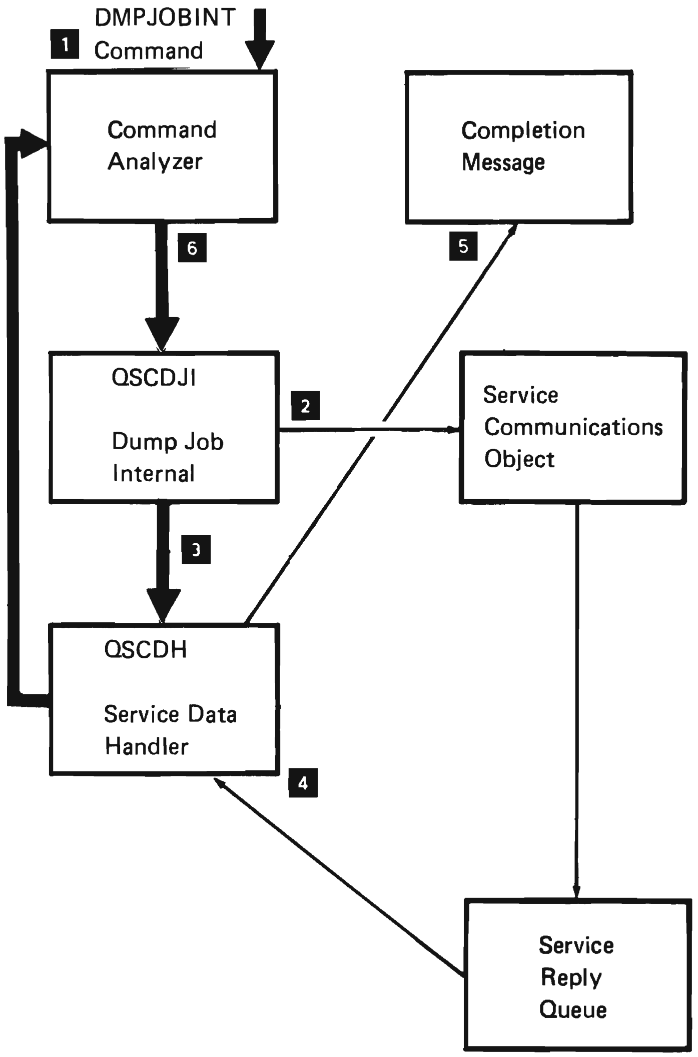


Figure SC-23. Dump Serviced Job Internal Command Overview

Dump Job Internal Command in Serviced Job

Figure SC-24 shows an overview of the Dump Job Internal (DMPJOBINT) command in a serviced job.

1 QSCGJI is called by QSCRH. The SCO is passed as input.

2 QSCGJI executes a diagnostic instruction to cause an internal process dump to be taken. The diagnostic instruction writes the dump to the internal vertical microcode log and returns the dump identifier to QSCGJI.

3 QSCGJI sends the dump identifier in the completion message to the servicing process by enqueuing it to the service data queue, and control is returned to the caller.

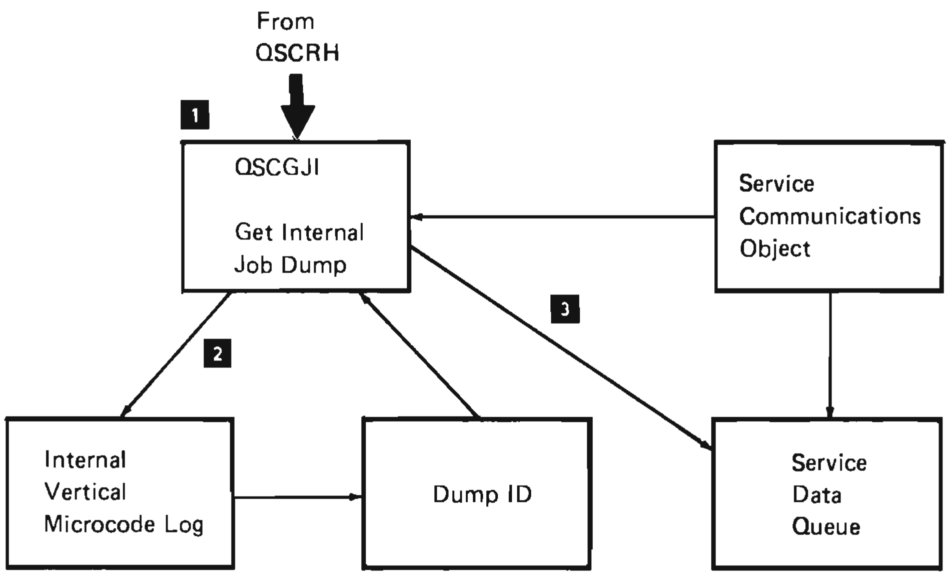


Figure SC-24. Dump Job Internal Command Overview

List Internal Data Command

Figure SC-25 shows an overview of the List Internal Data (LSTINTDTA) command.

1 The command analyzer decodes the List Internal Data command and control is transferred to QSCLID. The command parameter is passed as input.

2 QSCLID builds a command string using the input parameters. QSCLID invokes the internal macro ?CALLSF to establish a link to the concurrent service monitor.

3 The concurrent service monitor lists the requested internal data using REQIOs to the machine service component.

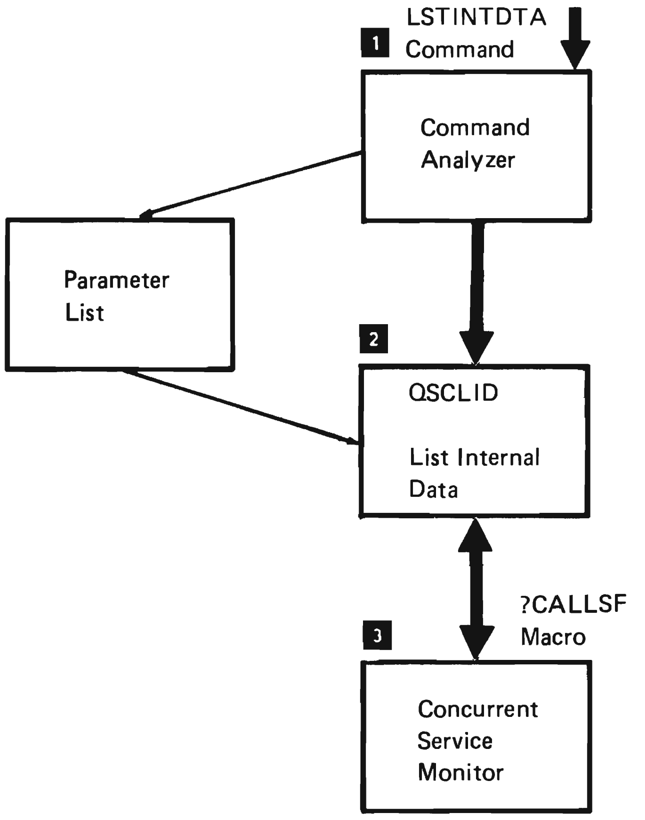


Figure SC-25. List Internal Data Command Overview

Display CSNAP Attributes Command

Figure SC-26 shows an overview of the Display CSNAP Attributes (DSPCNPA) command operation.

1 The command analyzer decodes the Display CSNAP Attributes command and control is passed to QSCDCNPA. The command parameter is passed as input.

2 QSCDCNPA builds a command string using the input parameter. QSCDCNPA invokes the internal macro ?CALLSF to establish a link to the concurrent service monitor.

3 The concurrent service monitor returns to QSCDCNPA the CSNAP short term statistics attributes set in the system. Then QSCDCNPA displays or lists the CSNAP short term statistics attributes set in the system.

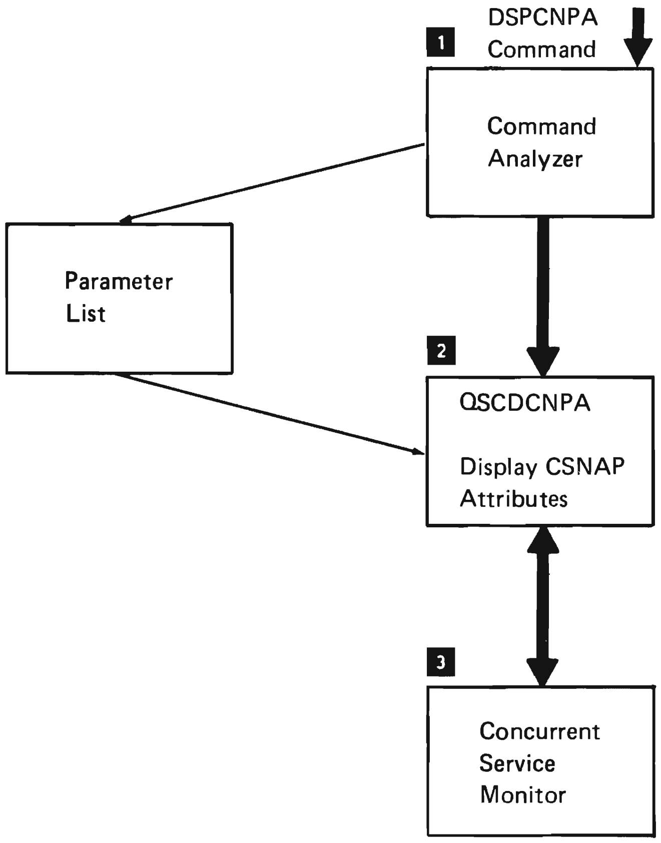


Figure SC-26. Display CSNAP Attributes Command Overview

Change CSNAP Attributes Command

Figure SC-27 shows an overview of the Change CSNAP Attributes (CHGCNPA) command operation.

1 The command analyzer decodes the Change CSNAP Attributes command and control is passed to QSCCCNPA. The command parameters are passed as input.

2 QSCCCNPA builds a command string using the input parameters. QSCCCNP invokes the internal macro ?CALLSF to establish a link to the concurrent service monitor.

3 The concurrent service monitor updates the appropriate CSNAP short term statistics attributes.

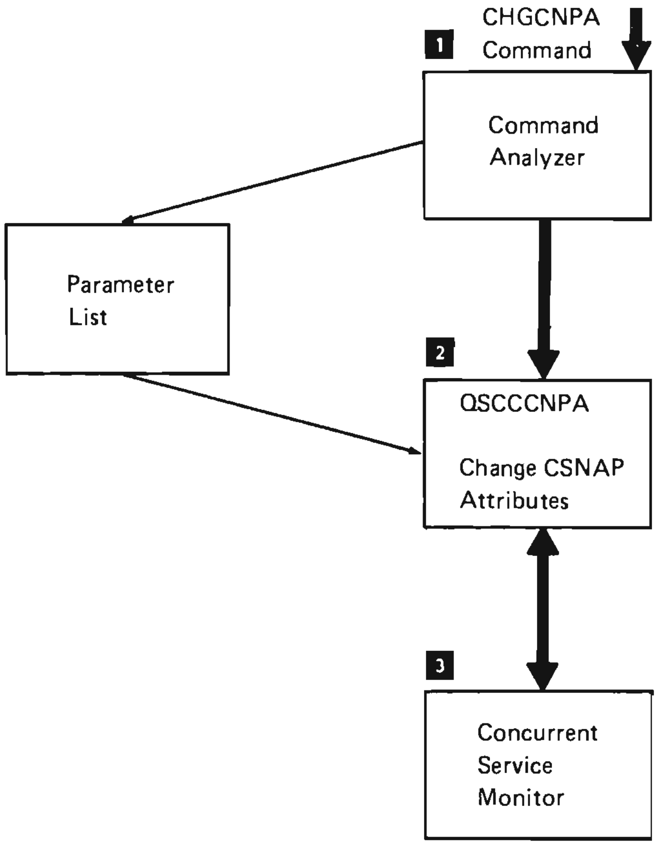


Figure SC-27. Change CSNAP Attributes Command Overview

List CSNAP Data Command

Figure SC-28 shows an overview of the List CSNAP Data (LSTCNPDTA) command operation.

1 The command analyzer decodes the List CSNAP Data command and control is passed to QSCLCNPD. The command parameter is passed as input.

2 QSCLCNPD builds a command string using the input parameter. QSCLCNPD then invokes the internal macro ?CALLSF to establish a link to the concurrent service monitor.

3 The concurrent service monitor lists the requested internal data using REQIOs to the machine service component.

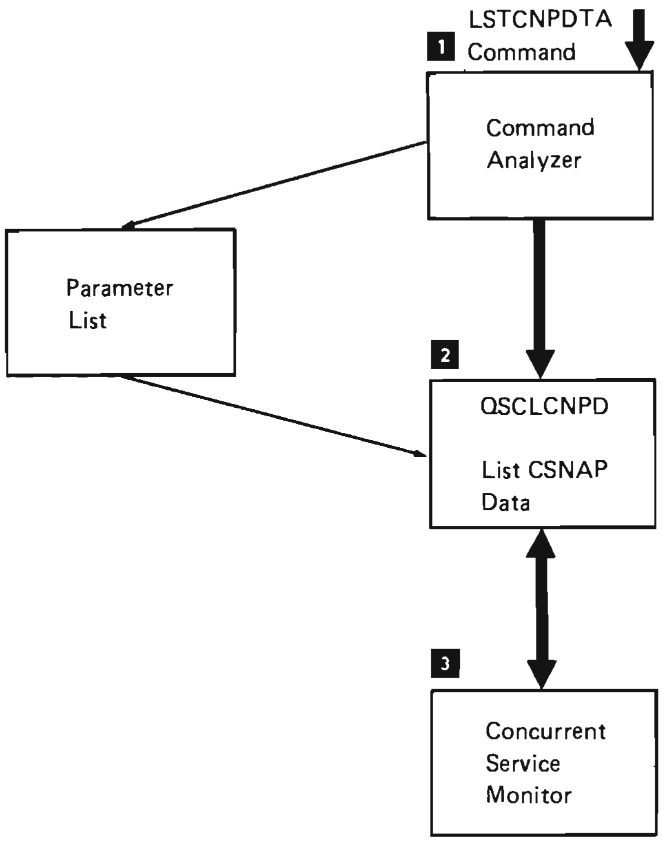


Figure SC-28. List CSNAP Data Command Overview

List CSNAP History Command

Figure SC-29 shows an overview of the List CSNAP History (LSTCNPHST) command operation.

1 The command analyzer decodes the List CSNAP History command and control is passed to QSCLCNPH. The command parameter is passed.

2 QSCLCNPH builds a command string using the input parameter. QSCLCNPH then invokes the internal macro ?CALLSF to establish a link to the concurrent service monitor.

3 The concurrent service monitor lists the requested internal data usin REQIOs to the machine service component.

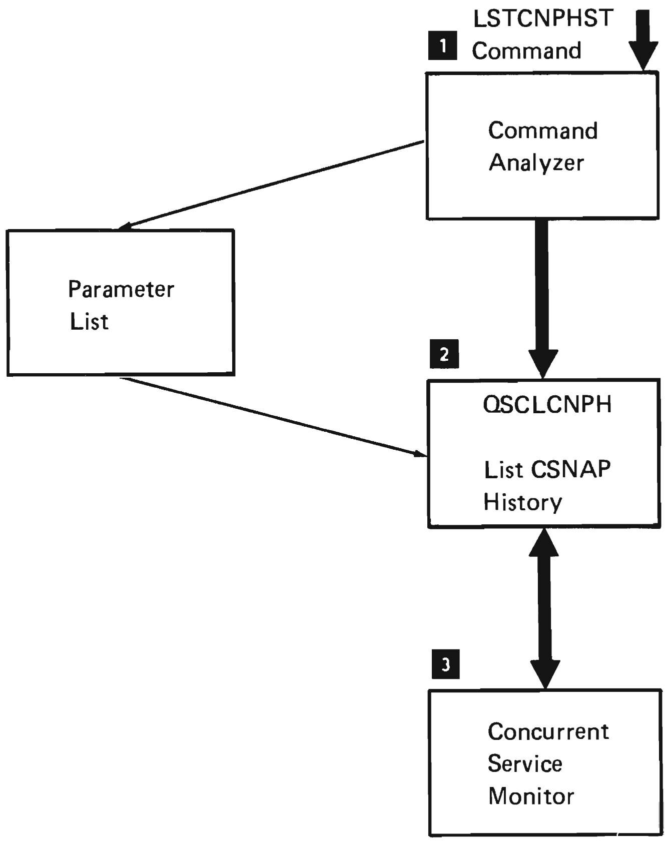


Figure SC-29. List CSNAP History Command Overview

Advanced Program-to-Program Communications Function Manager

INTRODUCTION

The APPC (advanced program-to-program communications) function manager component of CPF (control program facility) provides System/38 applications written in RPG III and COBOL a way to communicate using the SNA logical unit (type 6.2) protocols. The APPC function manager provides the interface between the application program and the APPC access path manager component. Using the SNA logical unit (type 6.2) protocols, the RPG III and COBOL applications programs running on the System/38 can communicate with an application program on another system.

The application program uses DDS (data description specifications) to describe the communications or mixed device files that are used to communicate between the System/38 and another device implementing SNA logical unit (type 6.2).

The following data management functions are supported by the APPC function manager:

- Open a file
- Acquire a device
- Send data
- Receive data
- Wait for receipt of data
- Release a device
- Close a file

The ?SIRVPIP and ?SIRSPAT macros are supplied as an ease-of-use interface to the access path component for use by the attach manager function.

GENERAL OVERVIEW

Advanced Program-to-Program Communications Function Manager Modules

The APPC function manager component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry into the component. Indentation of a module shows its dependency on a previous module.

-->QSICLOSE–APPC Close: This module is used to close a communications or mixed device file to input or output processing.

-->QSIEFEVH–Expedited Flow Unsolicited Data Processor: This module is called to process any expedited flow data received by the access path manager unsolicited data event handler.

> QSIERP1–I/O Error Processor: This module is used to process I/O errors returned to the APPC function manager.
>
> QSIERP2–Detected Error Processing: This module is used to build and send all messages for errors detected by the APPC function manager.
>
> QSIFMH7–Error Header Processor: This module is called when an error header (FMH-7) is received in the input buffer. QSIFMH7 performs all processing for the error header and any associated error log or error data.

-->QSIGET–APPC Get: This module handles the get data management macro functions for the APPC function manager. QSIGET receives data blocks from the access path component and passes them, one record at a time, to the user program. It performs the associated validity checks, deblocks the data using the record format retrieved from the device file, moves data to the users input buffer, and processes routing and response indicator keywords. QSIPUT transfers control to QSIGET to complete the processing of a put with invite function. QSIGET also performs the get portion of a put-get request.

QSIINASP–Initialize Associated Space: This module is used to initialize the associated space of QSIPUT.

-->QSIIOCMP–REQIO Complete Event Processor: This module is called by the access path component when it receives a REQIO complete event. QSIIOCMP is called to process a portion of a put with invite function. When a put with invite function is received, the APPC function manager requestes a REQIO complete event to be issued when the data is received. QSIIOCMP processes that event and issues a data available event when required.

-->QSINFEVH–Normal Flow Unsolicited Data Processor: This module is called to process all normal flow data received by the access path manager unsolicited data event handler.

-->QSIOPEN–APPC Open: This module opens a communications or mixed device file. The file can be opened for input, output, or input and output. As part of the open process, a request is made to the access path component to allocate a conversation.

-->QSIPUT–APPC Put: This module performs the put function for the put and put-get macro interfaces. Data is taken from the user buffer and moved to the output buffer. Communications functions requested by option keywords are performed. QSIPUT transfers control to QSIGET to complete the processing of a put with invite function, or to do the get portion of a put-get request.

QSIRSP–Respond to Request: This module is used to send a positive response, negative response, forward abort, or put failure from a send state. If a negative response or a forward abort is sent, QSIRSP also sends the error header and error data.

-->QSIRSPAT–Respond to Attach: This module is used to send a positive or negative response to the attach header (FMH-5).

-->QSIRVPIP–Receive Program Initialization Parameters: This module is used to receive the program initialization parameters data necessary to start a target program.

-->QSITPP–Target Termination Phase Program: This module handles the deallocating of a target conversation that is still allocated to a process when the job terminates.

-->QSIVRYOF–APPC Vary Off: This module is called at LUD vary off time to destroy the request block spaces, the assembly area space objects, and the associated queue.

-->QSIVRYON–APPC Vary On: This module performs the following functions when the LUD is varied on:
- Initializes the LUD-associated space
- Creates a queue to manage the REQIO blocks and record assembly areas
- Creates space objects for the record assembly areas used by the get function

Advanced Program-to-Program Communications Function Manager Operation

Figure SI-1 and the following text describe the operation of the APPC function manager.

1 QSIVRYON is called as part of the LUD vary on process.

2 A high-level language program, through the QDMCOPEN or QDMACQDV modules of common data management, calls QSIOPEN. QSIOPEN completes the open or acquire of a device file.

A An argument list is passed that contains a pointer to the UFCB (user file control block), and an index into the array of LUD pointers in the ODP (open data path). This argument list determines which files are opened to the device.

As part of the open or acquire request, the following functions are performed:

- The size of the user I/O buffer, allocated in the ODP, is determined.
- The ODP is extended, as necessary, to allocate a user I/O buffer or function manager work area.
- The function manager work area is initialized.
- An APCB (access path control block) is allocated for the file. The APCB is the interface to the access path manager.
- A request is made to the access path manager to allocate a conversation.

3 After the file has been opened, information may be sent to the remote program or received from the remote program. If this is a source program, QSIPUT may be called to start the desired program on the remote system.

A An argument list is passed that contains pointers to the UFCB, the option list, and the control list. The option list contains information as to whether the request is a put, get, or put-get. The control list indicates which record format in the communications or mixed device file should be used for this request.

QSIPUT will build an attach header and transmit the data to the remote system. The attach manager on the remote system will start the requested program. The remote program opens a communications or mixed file to the *REQUESTER device.

The source program may continue to issue put requests. QSIPUT will block the data into the transmit buffers. Two transmit buffers are used, one buffer will fill as the other buffer is transmitted. The buffer is transmitted when one buffer is filled and another put or get is performed, or when a keyword is issued that ends the data (ALWWRT, DETACH, FAIL, INVITE, CONFIRM). When the program requests data, the ALWWRT keyword is issued to cause the SNA change direction indicator to be sent.

B The message handler is called to signal exceptions.

4 After the file has been opened, information may be sent to the remote program or received from the remote program. If this is a target program, QSIGET may be called to receive the data.

A An argument list is passed that contains pointers to the UFCB, the option list, and the control list. The option list contains information as to whether the request is a put, get, or put-get. It also indicates whether QSIGET was called via the QDMACCIN module of common data management, as a result of a read by file request. The control list indicates which record format in the communications or mixed device file should be used for this request.

QSIGET will receive the data from the remote program and assemble the record in one of two data assembly areas. QSIGET uses one data assembly area as the user input buffer while assembling data in the other assembly area.

If QSIGET receives control because of a read to a specific device, QSIGET will deblock a record, place it in the user buffer (data assembly area), and return control to the user.

If QSIGET receives control from QSIPUT due to a put with invite request, QSIGET checks for a complete record. If a record is available, the CPF data available event is issued and control returns to the user. If no record is available, QSIGET issues a receive request to the access path component and returns to the user. When the receive request completes, QSIIOCMP is called by the access path component. If QSIIOCMP determines a complete record is available, the CPF data available event is issued.

If a read by file is issued, QDMACCIN issues a wait-on-event. This wait-on-event is satisfied by a CPF data available event. QDMACCIN calls QSIGET when it determines that the CPF data available event belongs to advanced program-to-program communications.

The user continues to issue read requests until the TRNRND response indicator is received.

B The message handler is called to signal exceptions to the user.

5 The remote system may send unexpected data to the local system when read requests are pending. When this occurs, the access path component receives an unsolicited data event. The access path component calls QSINFEVH or QSIEFEVH to process the data. The user is informed of the received information on the next read or write, depending on the type of information received.

6 After a communications or mixed device file has been processed, the application may issue a close file or release device. QDMCLOSE or QDMRLSDV will then call QSICLOSE to close the file.

A An argument list is passed that contains a pointer to the ODP control block, an index into the array of LUD pointers in the ODP that determines to which device the file is to be closed, and the type of close to perform. A temporary close is invalid for APPC. If a temporary close is encountered, an exception will be signaled.

If the conversation is in a send state, all buffered data will be transmitted. If this is a target program, the conversation may be reallocated with another open request; if this is a source program, a detach will be sent to end the conversation.

If the conversation is in a receive state and is a target program, no I/O is performed and the conversation can be reallocated with another open request. If in receive state and it is a source program, an SNA negative response is sent followed by an error header indicating program abend. A detach is then sent to end the conversation. For either target or source, the conversation will be deallocated.

B The message handler is called to signal exceptions to the user.

7 QSIVRYOF is called as part of the LUD vary off process.

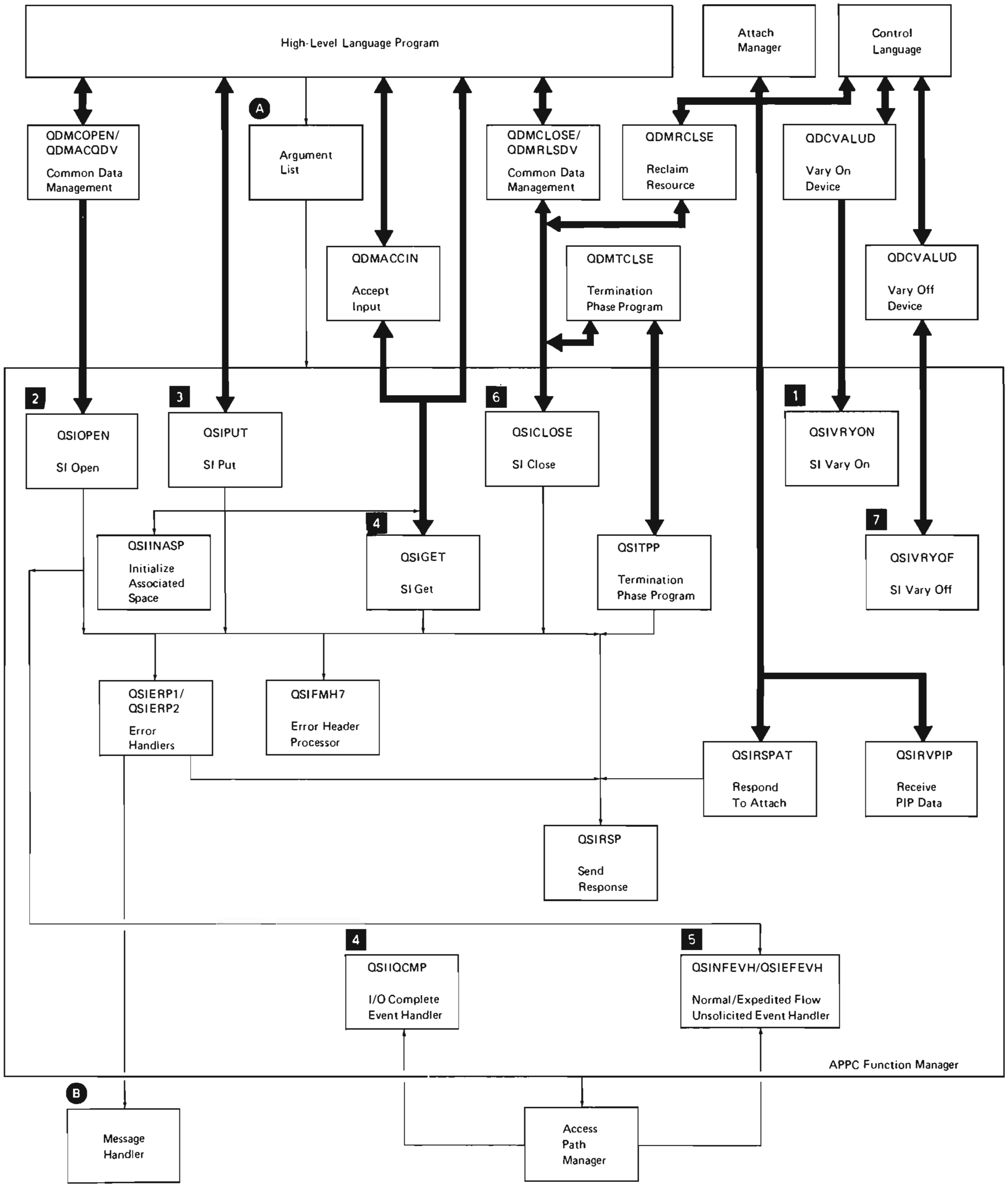

High-Level Language Program
Attach Manager
Control Language
QDMCOPEN/ QDMACQDV Common Data Management
A
Argument List
QDMCLOSE/ QDMRLSDV Common Data Management
QDMRCLSE Reclaim Resource
QDCVALUD Vary On Device
QDMACCIN Accept Input
QDMTCLSE Termination Phase Program
QDCVALUD Vary Off Device
2
QSIOPEN SI Open
3
QSIPUT SI Put
6
QSICLOSE SI Close
1
QSIVRYON SI Vary On
QSIINASP Initialize Associated Space
4
QSIGET SI Get
QSITPP Termination Phase Program
7
QSIVRYOF SI Vary Off
QSIERP1/ QSIERP2 Error Handlers
QSIFMH7 Error Header Processor
QSIRSPAT Respond To Attach
QSIRVPIP Receive PIP Data
QSIRSP Send Response
4
QSIIQCMP I/O Complete Event Handler
5
QSINFEVH/QSIEFEVH Normal/Expedited Flow Unsolicited Event Handler
APPC Function Manager
B
Message Handler
Access Path Manager

Figure SI-1. APPC Function Manager

Secondary Logical Unit

INTRODUCTION

The LU-1 (secondary logical unit type 1) component of CPF (control program facility) provides a means by which the System/38 is able to communicate with a primary logical unit (type 1). Using standard SNA protocols, RPG III and COBOL application programs running on the System/38 can communicate with application programs of a host system that uses CICS/VS or IMS/VS.

GENERAL OVERVIEW

Secondary Logical Unit Modules

The LU-1 component consists of the following modules:

Note: Modules identified by an arrow (-->) are entry modules into the component. Indentation of a module shows its dependency on a preceding module.

-->QSLOPEN–Open for LU-1: This module opens a communications file for input and output processing.

  QSLERBLD–Message Builder for LU-1: This module builds messages.

  QSLINLLS–Initializes the LU-LU SNA session and creates the GET/PUT request blocks.

-->QSLPUT–Put for LU-1: This module writes data to a communications file. It performs the associated validity checks, blocks the data using the record format retrieved from the device file, and processes keywords. It also performs the put portion of a put-get or a put with invite request.

  QSLERBLD–Message Builder for LU-1: This module builds messages.

-->QSLGET–Get for LU-1: This module retrieves records transmitted from the host system. It performs the associated validity checks, deblocks the data with the record format retrieved from the device file (based on the format selection type), moves the data to the user input buffer, and processes keywords. It also performs the get portion of a put-get or a put with invite request.

  QSLERBLD–Message Builder for LU-1: This module builds messages.

-->QSLCLOSE–Close for LU-1: This module closes a communications file to input and output processing.

  QSLERBLD–Message Builder for LU-1: This module builds messages.

-->QSLINASP–Initialization Routine for LU-1: This module initializes the LUD-associated space and creates indexes for SNA state machines.

-->QSLUSMON–Unsolicited Event Handler for LU-1: This module handles the unsolicited data events by calling QSLSNA to process the data. QSLUSMON also handles the REQIO complete events. Data transmitted from the host system is retrieved and then processed by calling QSLSNA. The data is deblocked with a record format retrieved from the device file, and based on the format selection type, moved to the user input buffer. When the record is complete, the CPF data available event is signaled.

    QSLSNA–SNA Protocol Enforcer for LU-1: This module provides the SNA control interface for the communication of data, commands, and responses, between the caller of this module and the communications device.

-->QSLSPEND–Suspends I/O Activity for LU-1: This module quiesces and suspends I/O activity for a previously opened device.

-->QSLRST–Restore I/O Activity for LU-1: This module restores I/O activity for a previously suspended device.

Secondary Logical Unit Operation

Figure SL-1 and the following text describe the operation of the LU-1 function manager.

1 A high-level language program, through the QDMCOPEN or QDMACQDV module of common data management, calls QSLOPEN to open a communications (LU-1) file for I/O processing, or acquire a device for a mixed file.

A An argument list is passed that contains a pointer to the UFCB (user file control block).

QSLOPEN calls QSLSNA to request that a session be initiated with the host system. QSLSNA issues all request I/Os necessary for session initiation:

- Receives BIND parameters, if available without sending LOGON text
- Sends LOGON text found in the communications file, if BIND parameters are not available
- Receives Bind command from host
- Receives Start Data Traffic command from host

B The message handler is called to signal exceptions to the user.

2 After the file has been opened, information may be transmitted to the host by calling QSLPUT.

A An argument list is passed that contains pointers to the UFCB, an option list, and control information. The option list contains information as to whether this request is a put or a put-get, and the control information indicates which record format in the communications file or mixed file should be used for this request.

When one of the put buffers becomes full, or a special function is requested through the use of communications file keywords, QSLPUT calls QSLSNA to issue a request I/O to the secondary station I/O manager to send data to the host system.

QSLPUT uses double buffering during write requests; one buffer is being filled while the other is being transmitted.

B The message handler is called to signal exceptions to the user.

3 After the file has been opened, information may be received from the host system by calling QSLGET or QDMACCIN.

A An argument list is passed that contains pointers to the UFCB, an option list, and control information. The option list contains information as to whether this request is a get or a put-get, and the control information indicates which record format in the communications file should be used for this request.

When the get buffer is empty, QSLGET calls QSLSNA to issue a request I/O to the secondary station I/O manager to receive data from the host system. If this is a put with invite request, QSLGET calls QSLSNA to issue a request I/O that causes an event to be signaled when the request I/O completes. If QSLGET can complete the request without calling QSLSNA, QSLGET will signal the CPF data available event. QSLGET passes the data to the application program one record at at time.

B The message handler is called to signal exceptions to the user.

4 The application program may issue a put-get or put with invite request. If this occurs, control is passed to QSLPUT.

After QSLPUT performs the put operation, QSLPUT transfers control to QSLGET for the get portion of the operation. QSLPUT does not regain control.

B The message handler is called to signal exceptions to the user.

5 The host system may, on occasion, send unexpected data or SNA command to the secondary logical unit while there is no read request pending. When that occurs, the secondary station I/O manager sends an event to the LU-1 function manager, causing QSLUSMON to be invoked. QSLUSMON calls QSLSNA to issue a request I/O to the secondary station I/O manager to receive the unsolicited data.

For put with invite requests, the request I/O issued causes an event to be signaled when the request I/O completes. This request I/O complete event invokes QSLUSMON. QSLUSMON then calls QSLSNA to process the data and deblock the data using a record format in the device file. If more data is required, QSLUSMON calls QSLSNA to issue another request I/O. When the record is complete, the CPF data available event is signaled.

6 After a communications file has been processed, QDMCLOSE or QDMRLSDV calls QSLCLOSE to close the file or release a device from a mixed file.

A An argument list is passed that contains pointers to the ODP (open data path), an index to the device being closed or released, and the type of close or release to perform. In this case, only a permanent close or release is valid; if a temporary close or release is specified, an exception will be signaled.

If any data remains in the put buffers at this point, QSLCLOSE calls QSLSNA to issue a request I/O to transmit this data to the host system. QSLCLOSE then recalls QSLSNA to request session termination. QSLSNA issues all request I/Os necessary for session termination:

- Sends Request Shut Down command
- Receives Unbind command from host system
- Sends LOGOFF text found in communications file

B The message handler is called to signal exceptions to the user.

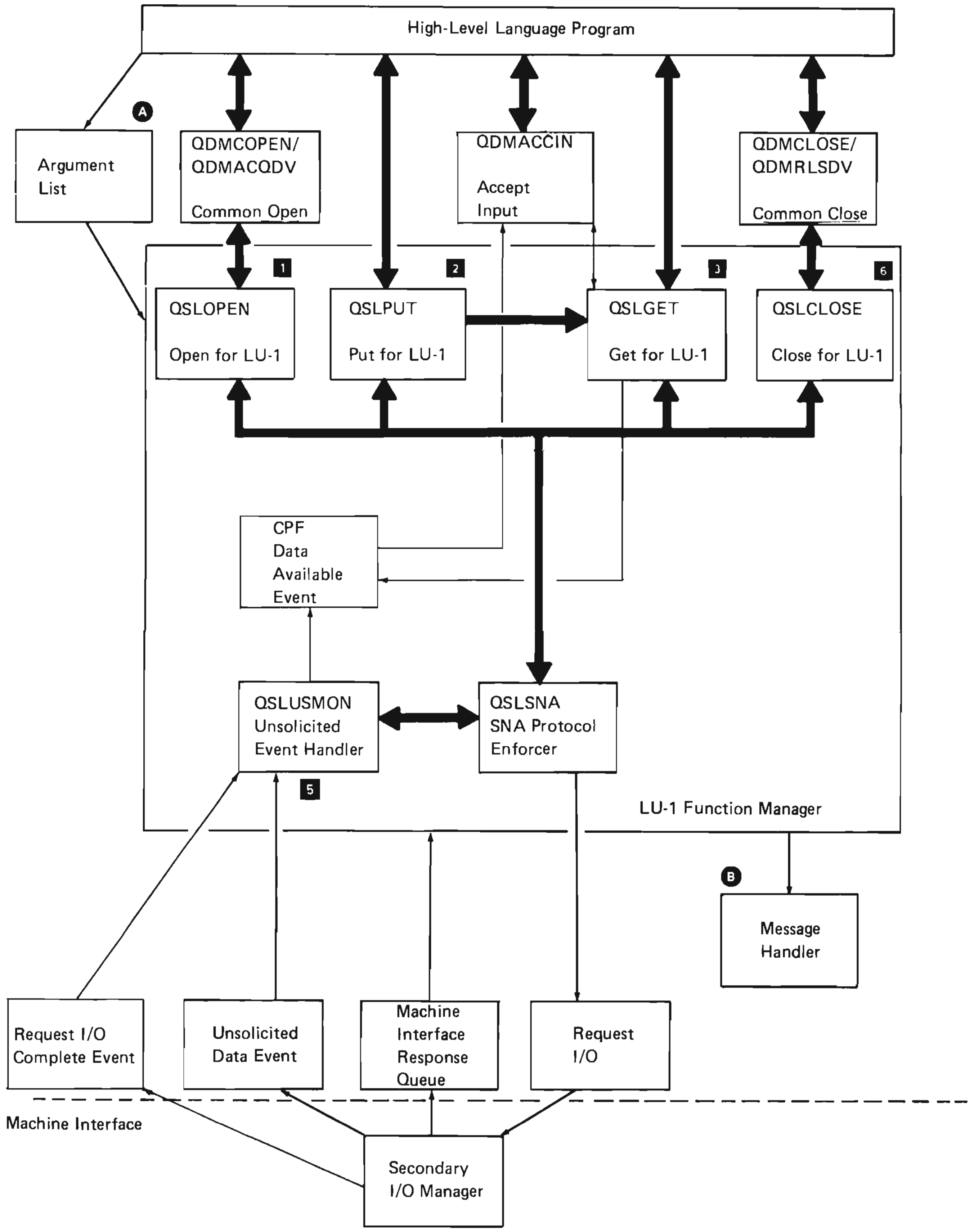


Figure SL-1. LU-1 Function Manager Operation Overview

INTRODUCTION

The CSM (concurrent service monitor) component of the CPF (control program facility) provides service personnel with an interface to various service functions. Through this interface, CSM handles requests and responses entered by service personnel from the operator console, as well as requests and responses from service functions (through the request I/O response queue).

CSM consists of three parts: initialization, response queue handler, and data available event handler.

Initialization

CSM is invoked as the result of a service person signing on the operator console. This starts the initialization process of the CSM, which provides the following functions:

- Validates CSM
- Creates and initializes CSM objects
- Establishes a service session with the service function driver
- Opens the operator console
- Displays the primary CSM menu

Response Queue Handler

Part of the CSM task is to process the requests of various service functions. A service function requests that CSM perform a function by enqueuing a message on the response queue. CSM then dequeues the response, handles the request, and notifies the service function of the results.

Data Available Event Handler

Another part of the CSM task is to process the requests of the service personnel using CSM. CSM presents a display to the user and then gives control to the response queue handler. When a service person responds to the display, control is given to the data available event handler, which processes the display, puts up the next display, and returns control to the response queue handler.

GENERAL OVERVIEW

Concurrent Service Monitor Modules

The CSM component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous level module.

-->QSMCCNLE–Control Cancel: This module handles the control cancel event.

-->QSMCSMSU–CSM Startup: This module receives control when CSM is invoked. It validates and initializes CSM.

    QSMSTART–Start CSM: This module creates and initializes the objects used by CSM and establishes a session with the service function driver.

    QSMCSMML–CSM Mainline: This is the controlling module when CSM is running.

        QSMRQDQR–Response Queue Dequeue Routine: This module dequeues and handles responses from the service functions.

        QSMDIAGD–Devices and Diagnostic Mode: This module sends a message to the console for all system devices that are in diagnostics prior to actual sign-off from CSM.

QSMSTOP–Stop CSM: This module performs a cleanup operation and then destroys the CSM objects.

-->QSMDAE–CSM Data Available Event Handler: This module receives control when a service display has been responded to. It does the initial processing required to call other CSM modules to handle the response.

QSMSFDR–Service Function Display Response Handler: This module processes the response to the service function displays.

QSMPSMR–Primary Menu Response Handler: This module processes the response to the primary menu.

QSMSFMR–Start Service Function Menu Response Handler: This module processes the response to the start service function menu.

QSMPSPR–Printer Selection Prompt Response Handler: This module processes the response to the printer selection prompt.

QSMASFDR–Active Service Function Prompt Response Handler: This module processes the response to the active service function prompt.

QSMSUIR–Service Utility Interface Display Response Handler: This module processes the response to the service utility interface display.

QSMVPUTY–Vary/Power Utility: This module processes the vary/power displays.

QSMELUTY–Error Logging Utility: This module allows service personnel to specify error logging for all errors or for threshold level errors where applicable.

The following CSM modules are used by other CSM modules to perform specific functions:

QSMREQIO–Request I/O: This module is used to initialize and issue the service request I/O instruction.

QSMDPM–Display Pending Message: This module presents the service displays to the service personnel.

QSMADDSF–Add a Service Function Area: This module adds a service function area to the CSM status area chain of active functions.

QSMDELSF–Delete a Service Function Area: This module deletes a service function area from the CSM status area chain of active service functions.

QSMCFKEY–Command Function Key Table Manager: This module handles the asynchronous command function key table.

QSMAVPR–Allocate, Vary, and Power Devices: This module is used to allocate, vary, and power devices as requested by the service person or service function.

QSMSFEX–Service Function End Exception Handler: This module handles the service function end exception.

QSMSSFX–Start Service Function Exception Handler: This module handles the start service function exception.

QSMOODE–Obtain Offline Device Event Handler: This module is an event handler that runs under control of the system arbiter process. It obtains offline devices for the CSM process.

Concurrent Service Monitor Initialization

Figure SM-1 and the following text describe the CSM initialization process.

1 The service person signs on the console with the assigned password. (CE is the password shipped with the system.) Because QSMCSMSU is the initial program specified in the CE profile, QSMCSMSU is invoked to validity check, initialize, and execute CSM.

2 QSMSTART is invoked to initialize CSM. A space object for the CSM status area is created and initialized. The CSM active bit is set on in the WCB (work control block). QSMREQIO is invoked to start the service session.

3 QSMDPM is invoked to present the primary menu.

4 Control is transferred to QSMCSMML (see Figure SM-2).

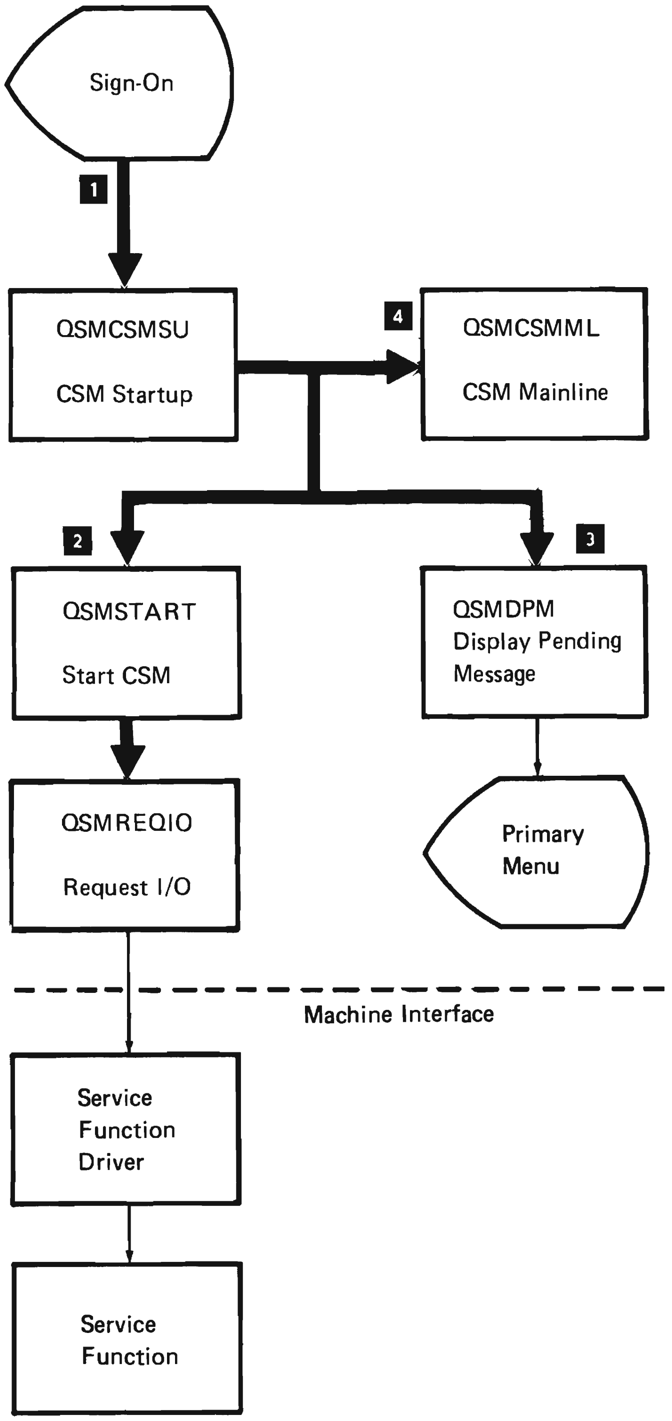


Figure SM-1. Concurrent Service Monitor Initialization Overview

Concurrent Service Monitor Response Queue Handling

Figure SM-2 and the following text describe the handling of messages placed on the response queue by the SFD (service function driver) and other service functions.

1 After the validity checking and initialization of CSM, control is transferred to QSMCSMML. The printer device files are overridden with a printer name that is not valid. This forces any printer that is opened to fail until the service person selects a printer.

2 QSMCSMML then calls QSMRQDQR to handle the response queue. QSMRQDQR dequeues responses from the response queue. If the response is:

- Unexpected error, CPF7204 exception is signaled.
- Service function is unable to start, the CPF7207 exception is signaled.
- Force return request from QSMDAE, control is returned to QSMCSMML.
- Service session stopped, control is returned to QSMCSMML.
- Service function destroyed, the service function is cleaned up and control is returned to QSMCSMML.
- Service function terminated, the completion code is checked. If the completion code is not zero, the CPF7206 exception is signaled. QSMREQIO is invoked to issue the destroy service function.
- Open display, a display I/O, or a close display, control is returned to QSMCSMML.
- Open data path, a data path I/O, or a close data path, control is returned to QSMCSMML.
- Open printer request, the printer is opened and QSMREQIO is invoked to notify the service function.
- Printer output request, the records are printed and QSMREQIO is invoked to notify the service function.
- Close printer response, the printer is closed and QSMREQIO is invoked to notify the service function.
- Open diskette request, the VTOC (volume table of contents) is opened and the buffer size is calculated based on the sector size. The diskette is opened and QSMREQIO is invoked to notify the service function.
- Diskette I/O request, the I/O request is performed and QSMREQIO is invoked to notify the service function.
- Close diskette request, the diskette is closed, the UFCB (user file control block) and I/O buffer space are destroyed, and QSMREQIO is invoked to notify the service function.
- Modify source/sink object request, addressability to the object is obtained and QSMAVPR is invoked to modify the object. QSMREQIO is invoked to notify the service function.
- Activate command function key request, QSMCFKEY is invoked to activate the command function keys. QSMREQIO is invoked to notify the service function.
- Deactivate command function key request, QSMCFKEY is invoked to deactivate the command function keys. QSMREQIO is invoked to notify the service function.
- Convert time request, QSMREQIO is invoked to notify the service function of the converted time.

When QSMCSMML regains control, the CSM status area is checked to see if a printer has been selected. If one has been selected, the printer device files are overridden. The reason that control was returned to QSMCSMML is checked. If the response is:

- Service function destroyed and there are no service function errors to display, QSMDELSF is invoked to clean up the service function objects.
- Open display, a space object is created to contain the display buffer. The address of the buffer is put into the service function status area and QSMREQIO is invoked to notify the service function.
- Display I/O, the pending display is put in the service function status area. If the display terminal is available, QSMDPM is invoked to present the display.
- Close display, the display space buffer object is destroyed and QSMREQIO is invoked to notify the service function.
- Data path open, I/O, or close, QSMREQIO is invoked to notify the service function of an error condition.
- Service session stopped, QSMDIAGD is invoked to display the devices that remain in diagnostic mode, and QSMSTOP is invoked to clean up the CSM objects and set off the CSM active bit in the WCB (work control block).

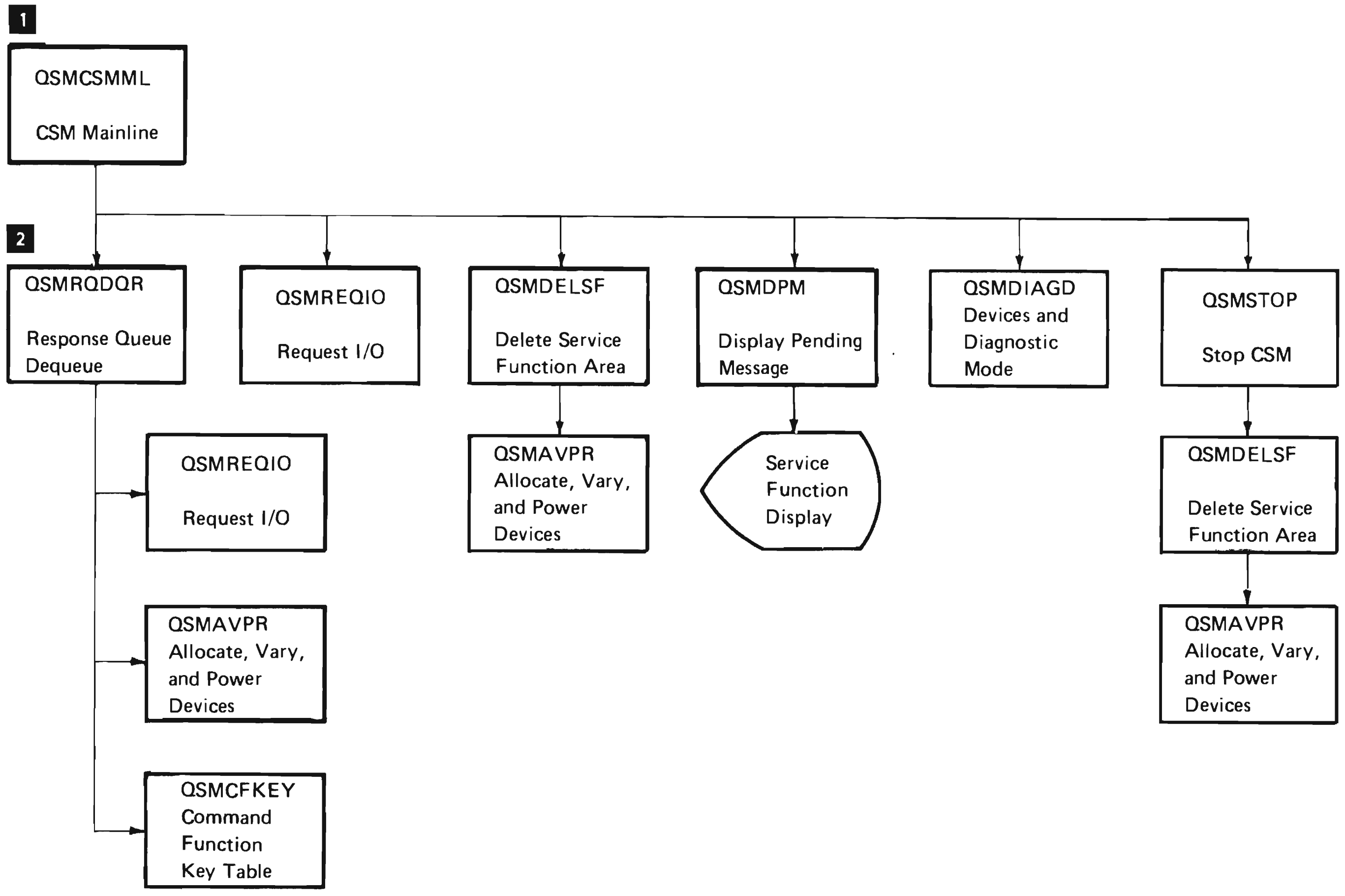


Figure SM-2. Concurrent Service Monitor Response Queue Handling Overview

Data Available Event Handler and Cancel Event Handler

Data Available Event Handler

Figure SM-3 and the following text describe what happens when the data available event is signaled and QSMDAE is invoked.

1 QSMDAE is invoked to process the data available event. Addressability to the CSM status area is resolved and the input data is obtained. One of the following then occurs:

- If the CF1 key was pressed, QSMDPM is invoked to present the primary menu.
- If the CF3 key was pressed, QSMDPM is invoked to erase and redisplay the display.
- If an asynchronous key was pressed, QSMCFKEY is invoked to process the key and then QSMDPM is invoked to redisplay the previous display.

2 If the display that was responded to was a service function display requiring input, QSMSFDR is invoked to process the response:

- QSMREQIO is invoked to notify the service function.
- QSMDPM is invoked to present the next display.
- If the display was a service function write-only display QSMDPM is invoked to present the next display.

3 If the display that was responded to was the primary menu, QSMPSMR is invoked to process the response and one of the following occurs:

- If the CF2 key was pressed, QSMDPM is invoked to present the primary menu.
- If the response was null, QSMDPM is invoked to resume interactive processing with an active service function, if one exists.
- If option 1 was selected, QSMREQIO is invoked to issue the stop service session request I/O.
- If option 2 was selected, QSMDPM is invoked to present the start service function menu.
- If option 3 was selected, QSMDPM is invoked to present the active service function prompt.
- If option 4 was selected, QSMDPM is invoked to present the active CF key display.
- If option 5 was selected, QSMDPM is invoked to present the printer selection prompt.
- If option 6 was selected, a space is created for the printer UFCB (user file control block), the printer is opened, and an indicator is set on in the CSM status area that indicates all displays are to be printed. QSMDPM is then invoked to present the next display.
- If option 7 was selected, the printer is closed, its UFCB space is destroyed, the print indicator in the CSM status area is set off, and QSMDPM is invoked to present the next display.
- If option 8 was selected, the interactive subsystem is started and QSMDPM is invoked to present the next display. If CSM is operating on an ISF system, the module QWCIINSR is destroyed to prevent any CPF start up attempt.
- If option 9 was selected, the command entry option is saved in the CSM status area, and QSMDPM is invoked to present the service utility interface display. The command entry screen is displayed after the response to the service utility interface display.

4 If the display that was responded to was the start service function menu, QSMSFMR is invoked to process the response and the following occurs:

- If the CF2 key was pressed, QSMDPM is invoked to present the primary menu.
- If the response was null, QSMDPM is invoked to resume interactive processing with an active service function if one exists.
- If display/alter/dump was selected, a message is sent to the history log to record that event.

- If service function was selected, QSMREQIO is invoked to issue the start service function request I/O.

- If a service utility was selected, the option selected is saved in the CSM status area and QSMDPM is invoked to present the service utility interface display.

5 If the display that was responded to was the active service function prompt, QSMASFDR is invoked to process the response and one of the following occurs:

- If CF2 was pressed, QSMDPM is invoked to present the primary menu.

- If the response was null, QSMDPM is invoked to resume interactive processing with an active service function if one exists.

- If the requested function was to cancel a service function, QSMREQIO is invoked to abort the service function.

- If the function requested was to select one service function for interaction, the interacting with all indicators in the CSM status area is set off and the service function to interact with is indicated.

- If the requested function was to select all service functions for interaction, the interacting with all indicators in the CSM status area is set on.

- QSMDPM is invoked to present the next display.

6 If the display that was responded to was the printer selection prompt, QSMPSPR is invoked to process the response and one of the following occurs:

- If the response was not valid, QSMDPM is invoked to redisplay the printer selection prompt with an error message.

- If the response was valid, the response is put into the CSM status area and a message is enqueued on the response queue. This forces QSMRQDQR to return to QSMCSMML (see Figure SM-2). QSMCSMML does an override of the device files to the printer name specified or to a spool file. The spool option is not presented or recognized on an ISF system. QSMDPM is then invoked to present the next display.

7 If the display that was responded to was the service utility interface display, QSMSUIR is invoked to start the service utility and one of the following occurs:

- If CF1 was pressed, QSMDPM is invoked to present the start service function menu.

- The service utility indicated in the CSM status area is invoked.

8 If the display that was responded to was the active command function key display or one of the CSM error displays, QSMDPM is invoked to resume interaction.

9 If QSMDAE function checks, then QSMDIAGD is invoked to display the device(s) in diagnostic mode prior to sign-off.

Control Cancel Event

10 When the control cancel event is signaled, QSMCCNLE receives control and calls QSMREQIO to issue the service request I/O instruction, which terminates each active service function. An indicator is set on in the CSM status area to prevent the service person from starting any new service functions.

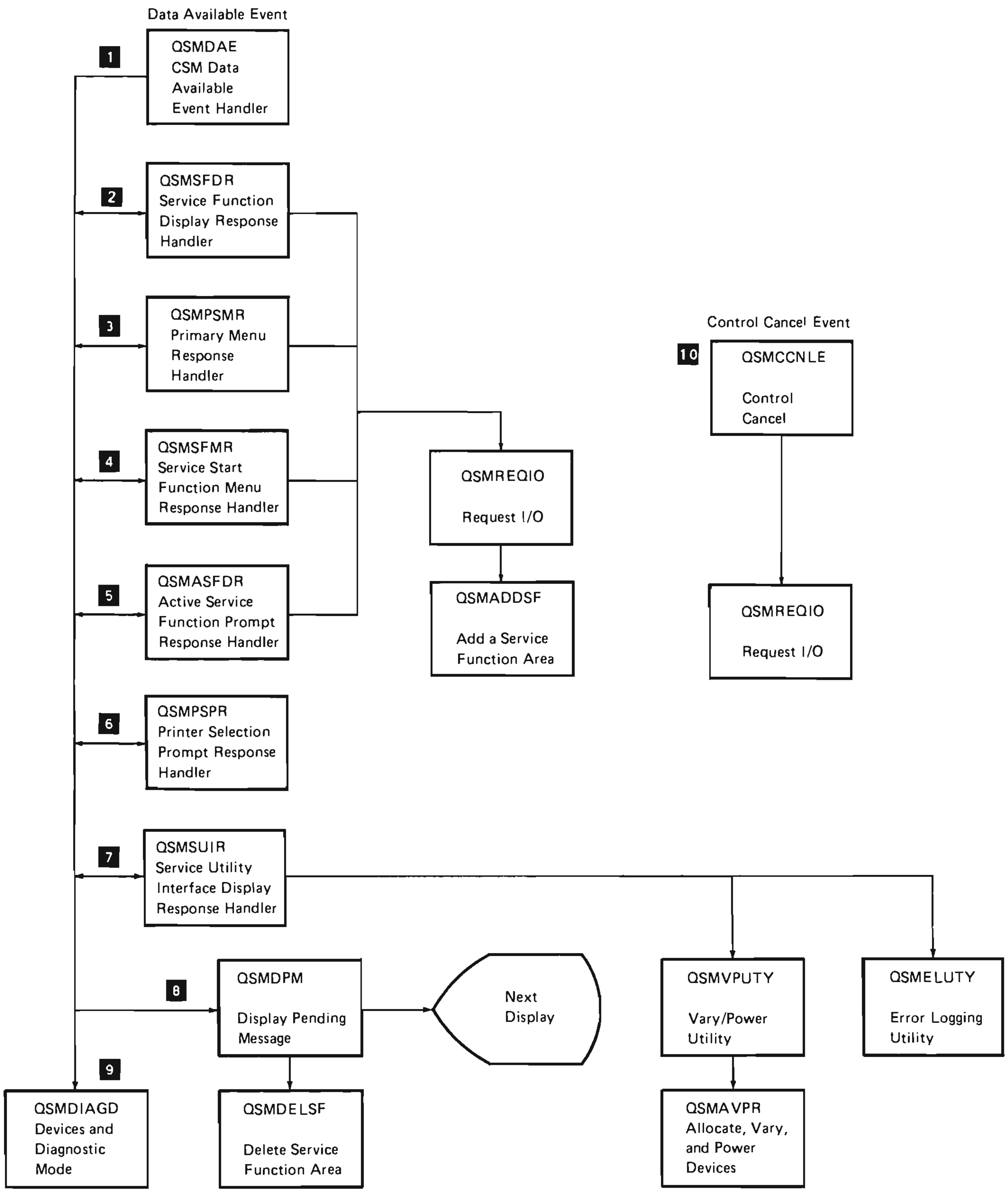


Figure SM-3. CSM Data Available Event Handler and Cancel Event Handler

Spooling

INTRODUCTION

The spooling component of the CPF (control program facility) is used to put batch job programs and their input data from a data base file member, a diskette, or the MFCU (multifunction card unit) into the system for processing. In turn, during processing the user can have printer, diskette, or MFCU output. The writer can then be started to a device (printer, MFCU, or diskette) for output from the output file.

Spooling Queues

A queue is an independent index. Each entry on the queue consists of a key portion and a data portion (job queue entries have only key portions). Individual queue entries are accessed by using the machine interface independent index support.

There are two types of queues: job and output.

Job Queue

The job queue is used to hold jobs that were entered into the system by a reader, a Submit Job (SBMJOB) command, a Submit Card Jobs (SBMCRDJOB) command, a Submit Data Base Jobs (SBMDBJOB) command, a Submit Diskette Jobs (SBMDKJOB) command, or jobs that have been moved by the Transfer Job (TRFJOB) command and are waiting to be executed. The job queues are used by the subsystems to handle the job requests.

Job queues can be manipulated by commands Clear Job Queue (CLRJOBQ), Hold Job Queue (HLDJOBQ), Release Job Queue (RLSJOBQ), Display Job Queue (DSPJOBQ), and Delete Job Queue (DLTJOBQ). Internally, job queues can be manipulated by macros that provide functions like put, delete, get highest priority job, and so forth. Job queues can be externally manipulated only by users that have the authority to use the job queues.

Output Queues

The output queue contains spooled output file entries created by an executing program that has specified spooling when the output file was opened. A writer, attached to the output queue, processes files as they become available.

Output queues can be manipulated and displayed by commands Clear Output Queue (CLROUTQ), Display Output Queue (DSPOUTQ), Hold Output Queue (HLDOUTQ), Release Output Queue (RLSOUTQ), Change Output Queue (CHGOUTQ), and Delete Output Queue (DLTOUTQ). Internally, output queues are manipulated by macros that provided functions such as put, delete, get output file, and so forth. Output queues can be externally manipulated only by users authorized to use the output queues.

GENERAL OVERVIEW

Spooling Modules

The spooling component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

Start CPF and Termination Modules

-->QSPSTCPF–Start CPF: This module, if requested to do so, clears queues at start CPF time. All job queues or output queues can be cleared.

-->QSPCLNUP–Clean-Up RWCB, Job Queues, and Output Queues: This module performs the following operations:

- Clears the RWCB (reader/writer control block), job queues, and output queues if the user specified *CLEAR for both the job queues and output queues on the start CPF prompt, or if the WCBT (work control block table) had been rebuilt by work control earlier in the IPL sequence.

- Clears from the RWCB entries for readers and writers that are not being left on a job queue through IPL.

- Calls QSPDAMGE if damage to job queues or output queues is detected; QSPDAMGE sends messages about the lost jobs or files (depending on the queue type), and destroys the queue.

- Removes entries from output queues for files that were being written and reentered into the system; changes the file status to not-being-written, and places undamaged entries back on the queue.

- Updates the status entry of each job queue to show no active subsystem, and updates the status entry of each output queue to show only writers waiting on a job queue during IPL.

- Grants authority to QSPL (spool profile) for each queue, if the spool profile was changed during the install.

-->QSPFFACB–Spooled Object Verify Routine: This module verifies the existence of the spooled objects, QSPL (library), QSPRWCB, QSPFACB, and data base file members. If they do not exist, QSPFFACB will create them. This module also clears the RWCB and resets all data base file members, if the user specified *CLEAR on the start CPF prompt for all job queues and all output queues, or if the WCBT was rebuilt. This module transfers object ownership of the spool objects to QSPL (spool profile) if the spool profile was changed during the install.

QSPDAMGE–Handle Damaged Objects: Performs the following operations:

- For damaged object QSPFACB, QSPDAMGE sends a lost job message for all jobs on the system, as well as a lost file message for every output file for the lost jobs.

- For damaged object QSPRWCB with readers and writers on a job queue, QSPDAMGE:
  - Removes the reader/writer entry from the job queue
  - Indicates to work control an active job in need of cleaning up
  - Sends the appropriate lost reader or writer message

- For each job on a damaged job queue, QSPDAMGE sends a lost job message, and indicates to work control an active job in need of cleanup.

- For each file on a damaged output queue, QSPDAMGE sends a lost file message, and reclaims the file member for reuse.

-->QSPTRMRW–Terminate Reader/Writer: This module is called by work monitor to clean up inline/output files and the RWCB entry, for reader/writer jobs in the termination phase.

Queue Command Modules

-->QSPCRSPQ–Create Queue (CRTJOBQ, CRTOUTQ)[1]: This module processes the Create Job Queue and Create Output Queue commands.

-->QSPCHSPQ–Change Queue (CHGOUTQ)[1]: This module processes the Change Output Queue command.

-->QSPHNSPQ–Delete/Move/Rename, Revoke Authority To Queue (DLTJOBQ, DLTOUTQ)[1]: This module processes the Delete Job Queue and Delete Output Queue commands. It provides special handling for Move Object, Rename Object, and Revoke Authority commands when they involve spooling queues.

-->QSPHLSPQ–Hold Job/Output Queue (HLDJOBQ, HLDOUTQ)[1]: This module processes the Hold Job Queue and Hold Output Queue commands.

-->QSPRLSPQ–Release Queue (RLSJOBQ, RLSOUTQ)[1]: This module processes the Release Job Queue and Release Output Queue commands.

-->QSPCLSPQ–Clear Queue (CLRJOBQ, CLROUTQ)[1]: This module processes the Clear Job Queue and Clear Output Queue commands and is also called from QSPSTCPF in the IPL sequence.

-->QSPDUPQ–Duplicate Job/Output Queue: This module provides handling for the Create Duplicate Object command when it involves spooling queues.

-->QSPSAVQ–Save Job/Output Queue: This module provides handling for the Save Object, Save Changed Objects, Save Library, and Save System commands when they involve spooling queues.

-->QSPRSTQ–Restore Job/Output Queue: This module provides handling for the Restore Object and Restore Library commands, and the install system procedure when they involve spooling queues.

Reader Function Modules

-->QSPSTRDR–Start Reader (STRCRDRDR, STRDBRDR, STRDKTRDR)[1]: This module processes the Start Card Reader, Start Data Base Reader, and Start Diskette Reader commands.

QSPFILES–Get Data Base File Member: This module provides a data base file member to the reader (for inline spooled files) or to spooling open (for spooled output) if a file member was not available in the FACB (file availability control block).

QSPRDR–Reader: This module uses the start reader command parameters to process the job stream.

QSPRDRAP–Spool Reader Adopt File: This module adopts the owner's authority and returns a pointer to the QSYS profile that contains that authority.

QSPDATAH–Reader Inline File Handler: This module reads into the system the inline files contained within a job.

-->QSPCTRDR–Control Reader Hold/Release (HLDRDR, RLSRDR)[1]: This module processes the Hold Reader and Release Reader commands.

-->QSPCNRDR–Cancel Reader (CNLRDR)[1]: This module processes the Cancel Reader command.

-->QSPSBMJB–Submit Jobs (SBMCRDJOB, SBMDBJOB, SBMDKTJOB)[1]: This module processes the Submit Card Jobs, Submit Data Base Jobs, and Submit Diskette Jobs command and reads the job stream from the appropriate device (or data base file).

QSPSBMEX–Submit Jobs Exit Program: This module provides error exit cleanup for QSPSBMJB.

-->QSPDFTCP–Default CPP (DATA, ENDJOB, and ENDINP): This module issues an escape when // is not found in front of the Data, End Job, or End Initial Process commands.

[1]This module is a CPP (command processing program).

Execution With Spooling Modules

-->QSPOPEN–Spooling Open: This module opens spooled output files to processing.

QSPCKPRM–Check Card and Diskette Open Parameters: This module is called to validate the card and diskette open parameters.

QSPIERRS–Issue Spool Intercept Error Messages: This module is used to send error messages from spool open and intercept modules.

-->QSPCLOSE–Spooling Close: This module is used to close spooled output files during the execution of a user program.

-->QSPFEOD–Force End of Data: This module performs a full close and open on a file.

-->QSPBPCRD–Spooling MFCU Put Intercept: This module is used to put intercepted MFCU records to a data base file member.

-->QSPBPPRT–Spooling Printer Put Intercept: This module is used to put intercepted print records to a data base file member.

-->QSPLSCPRT–Spooling SCS Print Intercept: This module intercepts SCS print commands and data, blocks them, and then calls QSPBPPRT to put the data to the intercept data base member.

-->QSPBPDSK–Spooling Diskette Put Intercept: This module intercepts user put diskette records, blocks them, and then puts the records to the intercept data base file member.

-->QSPRCMBR–Reclaim Spool Data Base Member: This module is used to make spool data base members available for reuse. It also releases job structures after the last spool file of a job is reclaimed.

Writer Function Modules

The functions of QSPPRTWT, QSPCRDWT, and QSPDKTWT are identical; only the devices are different. The functions consist of the following:

- Retrieve an available file from the output queue.
- Get data from the associated data base file member and put it to the device.
- If the AUTOTRM parameter on the Start Writer command is NO, wait on the spooled file available event when all available files are processed.
- If the AUTOTRM parameter is YES, transfer control to QSPWTRM2 at the appropriate time.

-->QSPSTWTR–Start a Writer (STRPRTWTR, STRCRDWTR, STRDKTWTR)[1]: This module processes the Start Printer Writer, Start Card Writer, and Start Diskette Writer commands.

QSPWTRM1–Writer Mainline Start: This module initializes the writer work space, allocates the device, addresses the output queue, and then transfers control to the appropriate writer module: QSPPRTWT for print, QSPCRDWT for card, and QSPDKTWT for diskette.

-->QSPWTREH–Stop Writer Event Handler: This module is invoked when the stop writer event is signaled by the hold spooled file command processing program, the cancel spooled file command processing program, or the hold writer command processing program. It sets indicator on or off in the writer work space to indicate if the current file being processed is to be held or canceled or if the writer is to be held.

QSPSCAN–Scan Spool Large Record: This module scans a spool large record to reconstruct the page profile environment of an enhanced function print file. This module is only used by QSPPRTWT.

-->QSPACCLG–Log Spool and Printer Job Accounting Data: This module logs the print job accounting data for the DP (direct print) and SP (spool print) entries to the QACGJRN.QSYS journal.

-->QSPHLWTR–Hold a Writer (HLDWTR)[1]: This module processes the Hold Writer command.

-->QSPRLWTR–Release a Writer (RLSWTR)[1]: This module processes the Release Writer command.

-->QSPCNWTR–Cancel a Writer (CNLWTR)[1]: This module processes the Cancel Writer command.

-->QSPCHWTR–Change a Writer (CHGWTR)[1]: This module processes the Change Writer command.

[1]This module is a CPP (command processing program.

Queue Management Module

-->QSPGPJOB–Get/Put/Delete Job: This module is used to get jobs from, put job entries into and delete job entries from a job queue.

Display Spool Data Modules

-->QSPDSPQ–Display Queue (DSPJOBQ, DSPOUTQ)[1]: This module processes the Display Job Queue and Display Output Queue commands.

-->QSPDSPJ–Display Job: This module displays the spool portion of the Display Job command.

-->QSPDSPF–Display Spooled File (DSPSPLF)[1]: This module processes the Display Spooled File Data command.

QSPDSPFX–Display Spooled File Exit Program: This module provides exit clean-up for QSPDSPF.

-->QSPDSPFA–Display Spooled File Attributes (DSPSPLFA)[1]: This module processes the Display Spooled File Attributes command.

QSPHNCMD–Handle Commands From Displays: This module processes the options from various displays to hold, release, cancel, or display specific files or jobs from spool displays that list files or jobs. This module also handles the messages associated with the execution of the display commands, and presents the messages on the updated display.

QSPPMTFA–Prompt with Spool File Attributes: This module produces a prompt of the CHGSPLA command with the current spool file attributes filled in as defaults.

-->QSPDSPRW–Display Reader/Writer (DSPRDR, DSPWTR)[1]: This module handles the RDR/WTR (*ALL) portion of the Display Reader and Display Writer commands. It produces a display of all readers or writers on the system, or transfers control to QSPDSPR or QSPDSPW for a display of a specific reader or writer.

-->QSPDSPR–Display Reader: This module handles the specific name portion of the Display Reader command, producing a display of the particular reader.

-->QSPDSPW–Display Writer: This module handles the specific name portion of the Display Writer command, producing a display of the particular writer.

Job/File Command Modules

-->QSPCPYF–Copy Spooled File (CPYSPLF)[1]: This module processes the Copy Spooled File command.

-->QSPHLSPF–Hold Spooled File (HLDSPLF)[1]: This module processes the Hold Spooled File command.

-->QSPRLSPF–Release Spooled File (RLSSPLF)[1]: This module processes the Release Spooled File command.

-->QSPCNSPF–Cancel Spooled File (CNLSPLF)[1]: This module processes the Cancel Spooled File command.

-->QSPCHSPF–Change Spooled File Attributes (CHGSPLFA)[1]: This module processes the Change Spooled File Attributes command.

-->QSPCNJOB–Cancel Job (Spooled Files): This module is called by work monitor to handle a portion of the Cancel Job command.

-->QSPCHJOB–Change Job (Spooled Files) EOJ, Hold, Release, or Change: This module is called by work monitor to handle a portion of the functions for the Hold Job, Release Job, and Change Job commands, as well as part of termination phase processing. It is also called by work control to handle a portion of IPL job clean-up.

[1]This module is a CPP (command processing program).

QSPCNVRT–Convert Printer/Diskette/Card Large Records: This module unblocks large records to individual print lines or records. This module is used by QSPCPYF and by the writers.

QSPCVDSP–Convert Printer/Diskette/Card Large Records: This module unblocks large records to individual print lines or records. This module is used by QSPDSPF.

QSPCVPRT–Convert SCS Print Records: This module unblocks large records to create individual print lines preceded by either a first character forms control character or a 4-character control code. This module is used by QSPCPYF and the print writer.

Spooling a Jobstream Into the System

Figure SP-1 and the following text describe what happens when a start reader command, which executes in the interactive subsystem, is entered from the console.

1 A start reader command (STRCRDRDR, STRDBRDR, or STRDKTRDR) is entered from the console. The command invokes QSPSTRDR, which puts a job entry on the appropriate job queue (spool subsystem queue).

2 An entry for the specified enqueued reader is put into the RWCB (reader/writer control block) along with the required reader input parameters.

3 The spool subsystem monitor then requests a job from its queues and allocates a temporary job structure. It then creates the reader process as specified by the user.

4 The reader process begins reading from the appropriate input device and obtains a complete JOB statement (job A).

5 The reader invokes the command analyzer to validity check the JOB statement.

A If the JOB statement is not valid, the reader receives control and puts the job in error to a specially created error job to hold all CL commands until a valid JOB statement is found.

B If the JOB statement is valid, the Command Analyzer transfers control to the proper command processing program to process the Job command.

C The command processing program allocates a permanent job structure: WCBT (work control block table entry), JMQ (job message queue), and SCB (spooling control block).

6 After the JOB command processing program completes its function, control is returned to the reader, which sends the CL associated with the JOB statement previously processed to the external message queue of the new job (part of its job message queue) as requests.

D If syntax checking was specified on the JOB statement, each CL command is sent to the command analyzer to be validity and syntax checked. All commands are put on the external message queue of its new job as requests.

E If inline data is encountered, the reader obtains a data base file member (XX) for the data and puts the file and member names (XX) into an inline control block entry in the SCB of the job. Named files contain the name provided on the DATA statement. Unnamed files contain the name QINLINE, which are put in the SCB in the order that they were encountered, and are made available for use by the user program.

7 On completion of the input, the reader invokes the spool queue management, which puts an entry for the job on the appropriate job queue. If the new batch job that was just put on the job queue is available for execution and if a subsystem is attached to that same queue, an event is signaled to the subsystem telling it that a batch job is available for selection to execute.

8 If Job A were found to be nonexecutable (an error of severity higher than the maximum allowed), the reader produces the joblog for Job A.

At the completion of a job, the reader returns to step 4 to read in the next job. The reader will continue to repeat this cycle until end-of-file or the reader is canceled. Note that the reader is only active for data base and magnetic media devices until end-of-file but is active for the MFCU until the reader is specifically canceled.

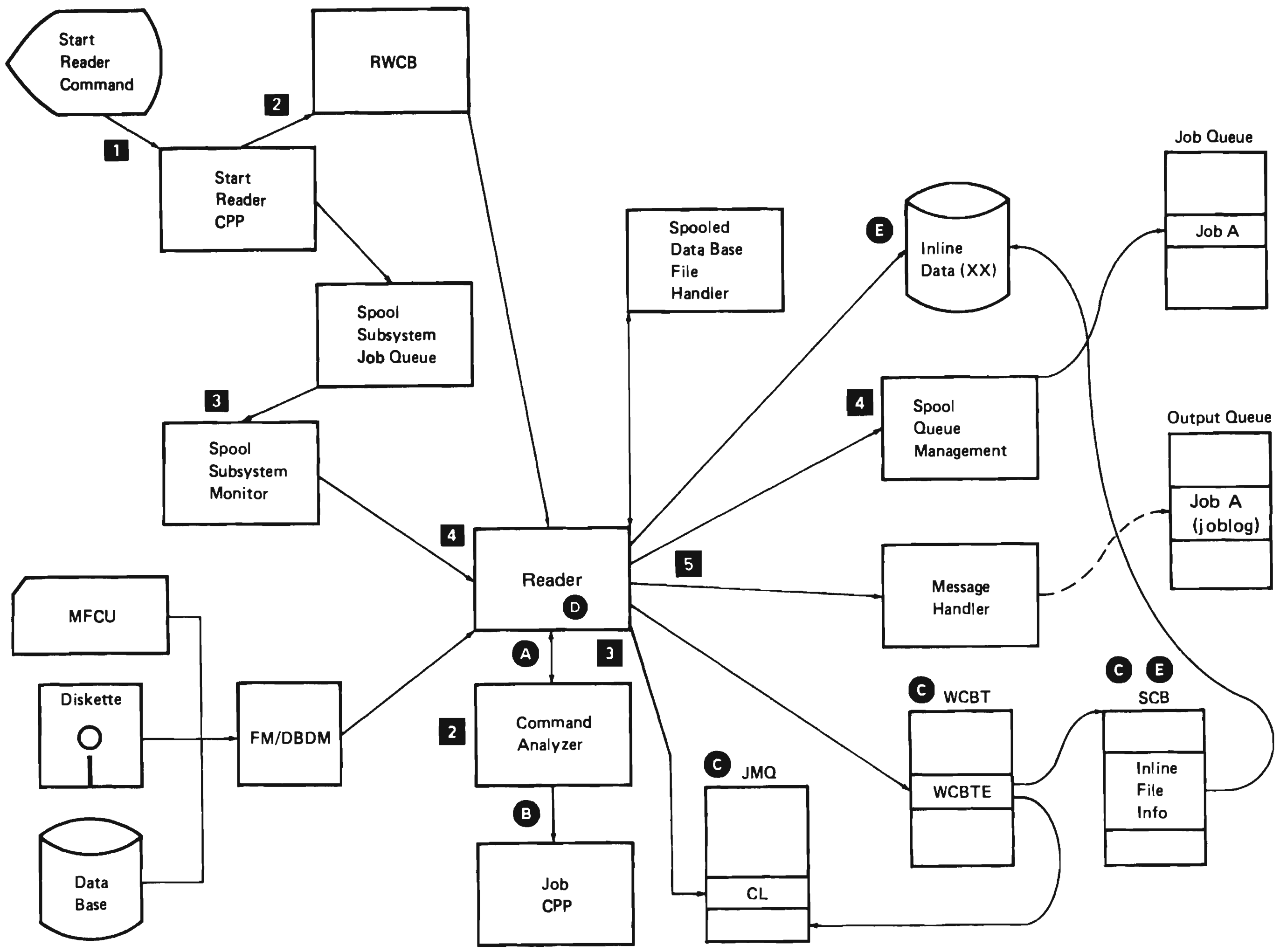


Figure SP-1. Spooling a Jobstream into the System

Spooling a Jobstream Into the System Using a Submit Jobs Command

Figure SP-2 and the following text describe what happens when a submit jobs command, which executes in the interactive subsystem, is entered from the console.

1 A submit jobs command (SBMCRDJOB, SBMDBJOB, or SBMDKTJOB) is entered from the console. The command invokes QSPSBMJB, which begins reading from the appropriate input device and obtains a complete JOB statement (job A).

2 QSPSBMJB invokes the command analyzer to validity check the JOB statement.

A If the JOB statement is not valid, the submit jobs command processing program receives control and puts the job in error to a specially created *error job* to hold all CL commands until a valid JOB statement is found.

B If the JOB statement is valid, the command analyzer transfers control to the job command processing program to process the Job command.

C The job command processing program allocates a permanent job structure: WCBT (work control block table) entry, JMQ (job message queue), and SCB (spooling control block).

3 After the Job command completes its function, control is returned to the submit jobs command processing program, which sends the CL associated with the JOB statement previously processed to the external message queue of the new job (part of its job message queue) as requests.

D No syntax checking will be performed on the submit jobs commands, except for // commands. All commands are placed on the external message queue of their new jobs.

E If inline data is encountered, the submit jobs command processing program obtains a data base file member (XX) for the data and puts the file and member names (XX) into an inline control block entry in the SCB of the job. Named files contain the name provided on the DATA statement. Unnamed files contain the name QINLINE, which are put in the SCB in the order that they were encountered, and are made available for use by the user program. (See Figure SP-4.)

4 On completion of the input, the submit jobs command processing program invokes the spool queue management, which puts an entry for the job on the appropriate job queue. If the new batch job that was just put on the job queue is available for execution and if a subsystem is attached to that same queue, an event is signaled to the subsystem telling it that a batch job is available for selection to execute.

5 If Job A were found to be nonexecutable (an error of severity higher than the maximum allowed), the submit jobs command processing program produces the joblog for Job A.

At the completion of a job, the submit jobs command processing program attempts to read in the next job. The command processing program will continue to repeat this cycle until the end-of-file. Note that the command processing program is only active for data base and magnetic media devices until EOF (end-of-file) but is active for the MFCU until the job/request is specifically canceled or until the End Initial Process command is encountered in the jobstream.

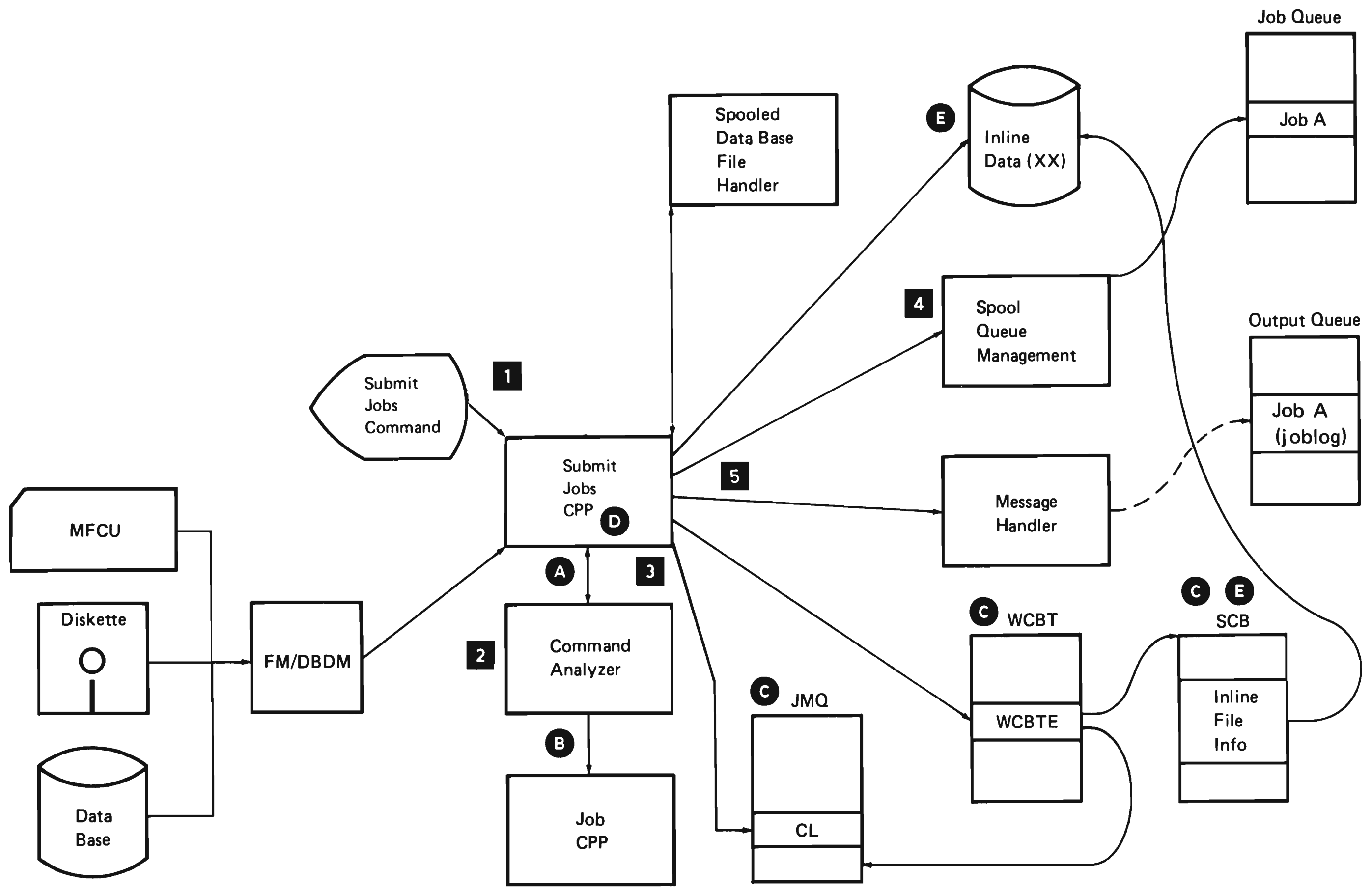


Figure SP-2. Spooling a Jobstream into the System Using a Submit Jobs Command

Job Selection by Work Monitor

Figure SP-3 shows an overview of how a job is selected for execution by work monitor.

After the job is placed on a job queue and when the available job event is signaled by spooling job queue management, the monitor attached to that job queue (if a new job can be scheduled) calls spooling queue management to get the next job scheduled for execution. In order to get the next job, the job queue and the job itself must not be in a held state. Work monitor makes two calls to the spooling queue management to get the next job.

1 First Call–The first call gets a pointer to the WCBTE (work control block table entry) of the selected job.

- **A** The first call request from work monitor is to select an available job.
- **B** The response to work monitor after the first call is a pointer to the selected job's WCBTE. The job queue entry is removed.

2 Second Call–If the selected job has any inline files, the override statements are built in the DMCQ (data management communioations queue) of the job such that when the job begins to execute and opens one of the inline files, common open can find the real data base file member to open for the program. The override statement in the DMCQ does the mapping of the inline file name of the user or QINLINE (the name assigned by spooling for unnamed data files) to the file member where spooling put the inline data records.

- **C** The request of the second call from work monitor is to build into the DMCQ of the job structure any overrides that are needed by inline data files.
- **D** The response to work monitor after the second call implies that all work was done for all inline data files in the job.

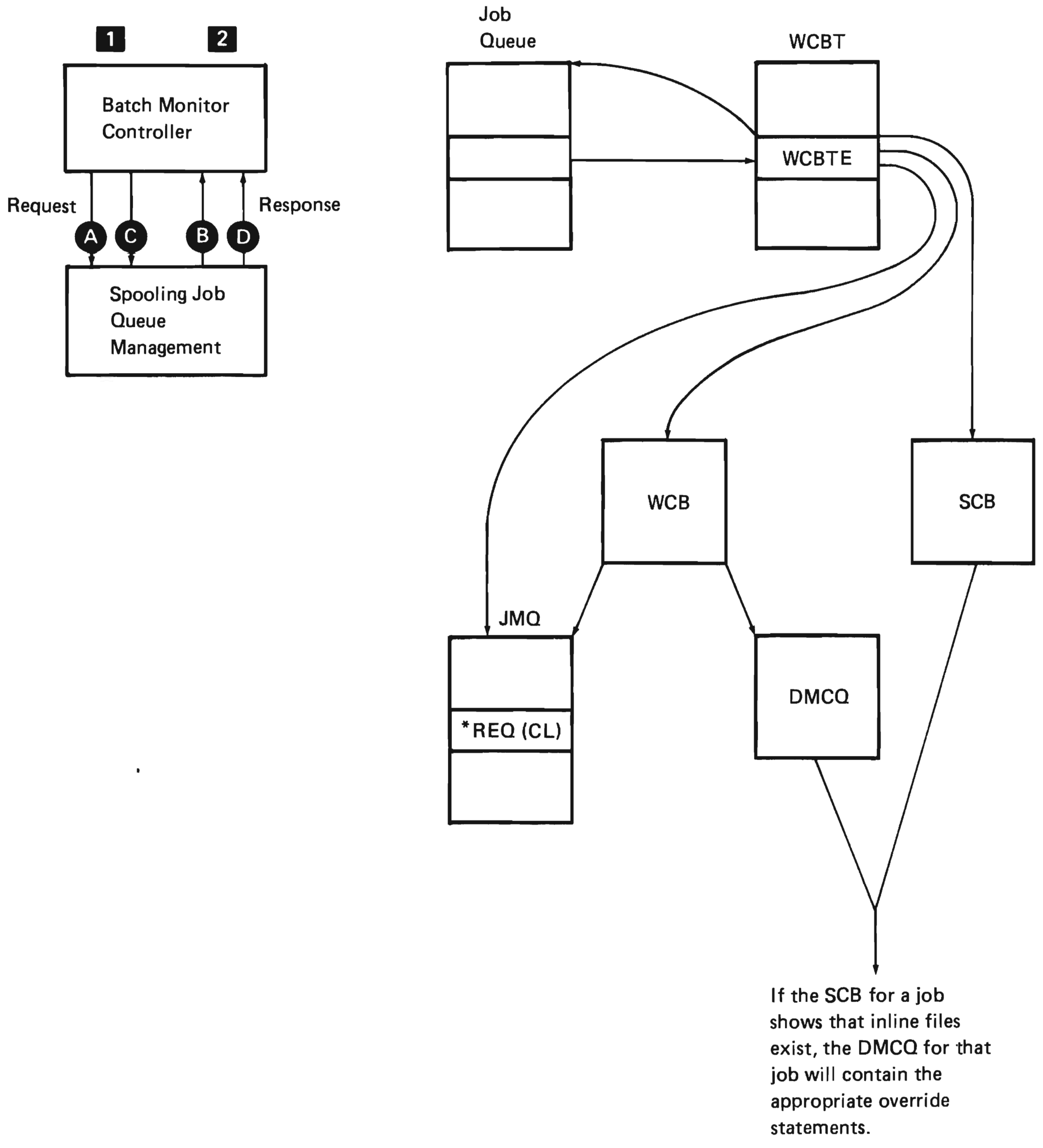

1
2
Batch Monitor Controller
Request
Response
A
C
B
D
Spooling Job Queue Management
Job Queue
WCBT
WCBTE
WCB
SCB
JMQ
*REQ (CL)
DMCQ
If the SCB for a job shows that inline files exist, the DMCQ for that job will contain the appropriate override statements.

Figure SP-3. Job Selection by Work Monitor

Executing a Program That Receives Spooled Inline Files

Figure SP-4 and the following text describe the operation of a program that receives spooled inline files.

If an inline data file is opened during execution, one of the following occurs:

Processing of Named Inline Data Files

The QDMCOPEN module of common data management is invoked to perform an open of a named inline data file. It looks on the DMCQ to locate a message for the named file. Finding the message, it obtains a pointer to the data base file member. For named inline data files, common data management does not remove the message from the DMCQ. QDMCOPEN invokes the data base open to process the named spooled file. Named spooled inline files can be reused.

Processing of Unnamed Inline Data Files

The QDMCOPEN module of common data management is invoked to perform an open of an unnamed inline data file. It looks on the DMCQ to locate a message for the QINLINE file. Finding the message, it obtains a pointer to the data base file member. For unnamed inline data files, common data management removes the message from the DMCQ. QDMCOPEN invokes the data base open to process the unnamed spooled file. Unnamed spooled inline files cannot be reused.

Data base get support is used to obtain records from the data base file for inline files. QDMCLOSE is used to close both unnamed and named inline files.

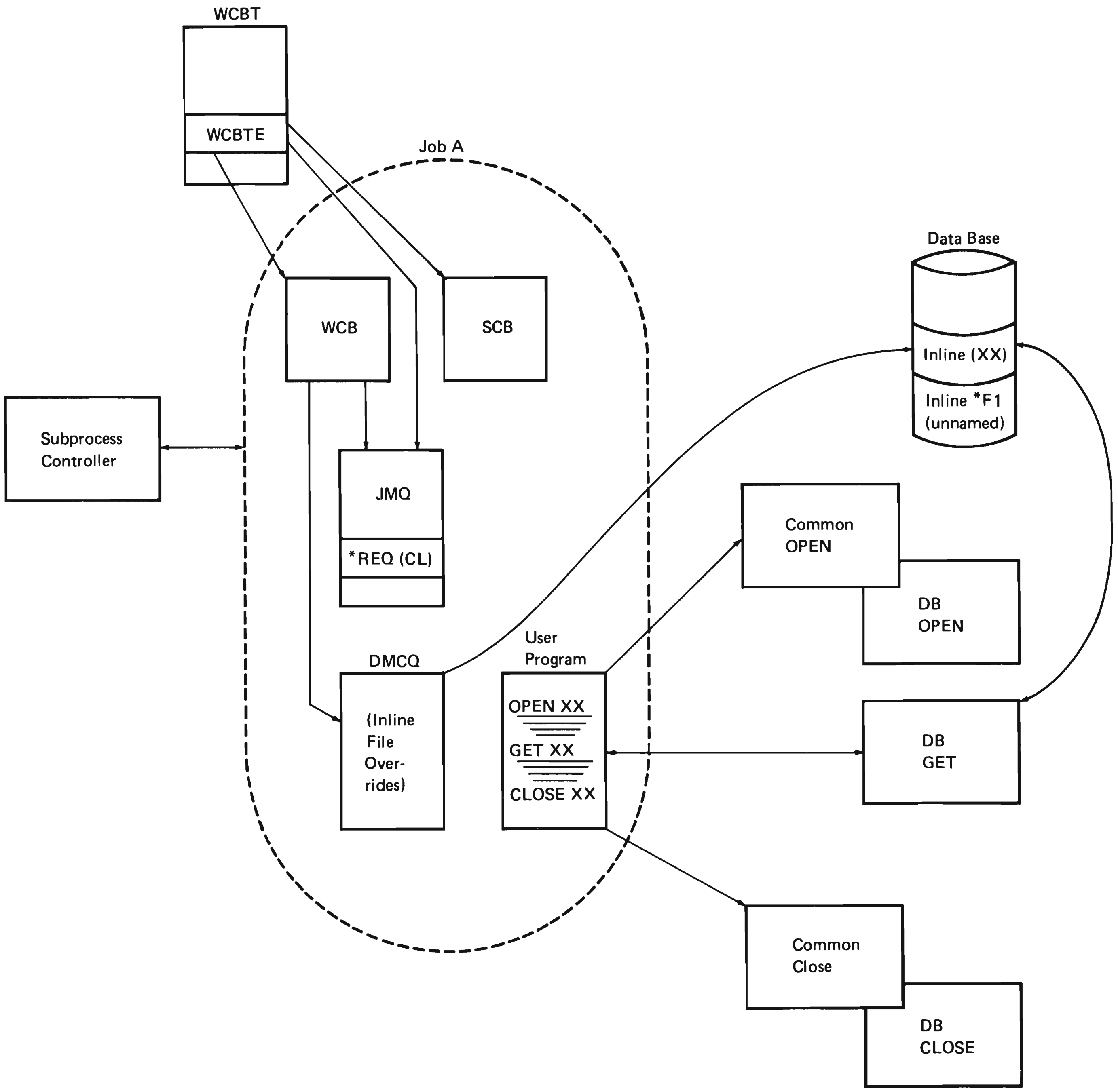


Figure SP-4. Executing a Program that Receives Spooled Inline Files

Executing a Program That Produces Spooled Output

Figure SP-5 and the following text describe the operation of a program that produces spooled output.

When QDMCOPEN opens an output device file with the parameter SPOOL=YES, it calls spooling open. Spooling open obtains a spooled data base file member to hold the spooled output of the program. Spooling open will put into the SCB (spooled control block) of the job the following:

- OFCB (output file control block)
- Writer-time UFCB
- Update the SCB to reflect that another spooled file belongs to the job

The appropriate output queue receives a new entry for this spooled output file. Then spooling open will open the data base file member used to store intercepted records. The application program (job J) puts records, using the spooling put intercept routines that block the data (see *Spooled Large Records*), to the spooled data base file member.

Upon completion of the user spooled output file, spooling close marks the file complete and available (if not already marked), and if the file is to be available before end-of-job for output on the spooling output queue. Spooling close closes the data base file member.

If the program closes the spooled file with a temporary close, the following events happen:

- The current large record is put to a data base file member.
- The next open will not let spooling get a new data base file member but will let new records be added to this file.

If FEOD (forced end-of-data) is used by the program the following events happen:

- The current data base file or member is closed.
- The same device file is used to open a new spooled file, creating a new data base file member for intercepted records.

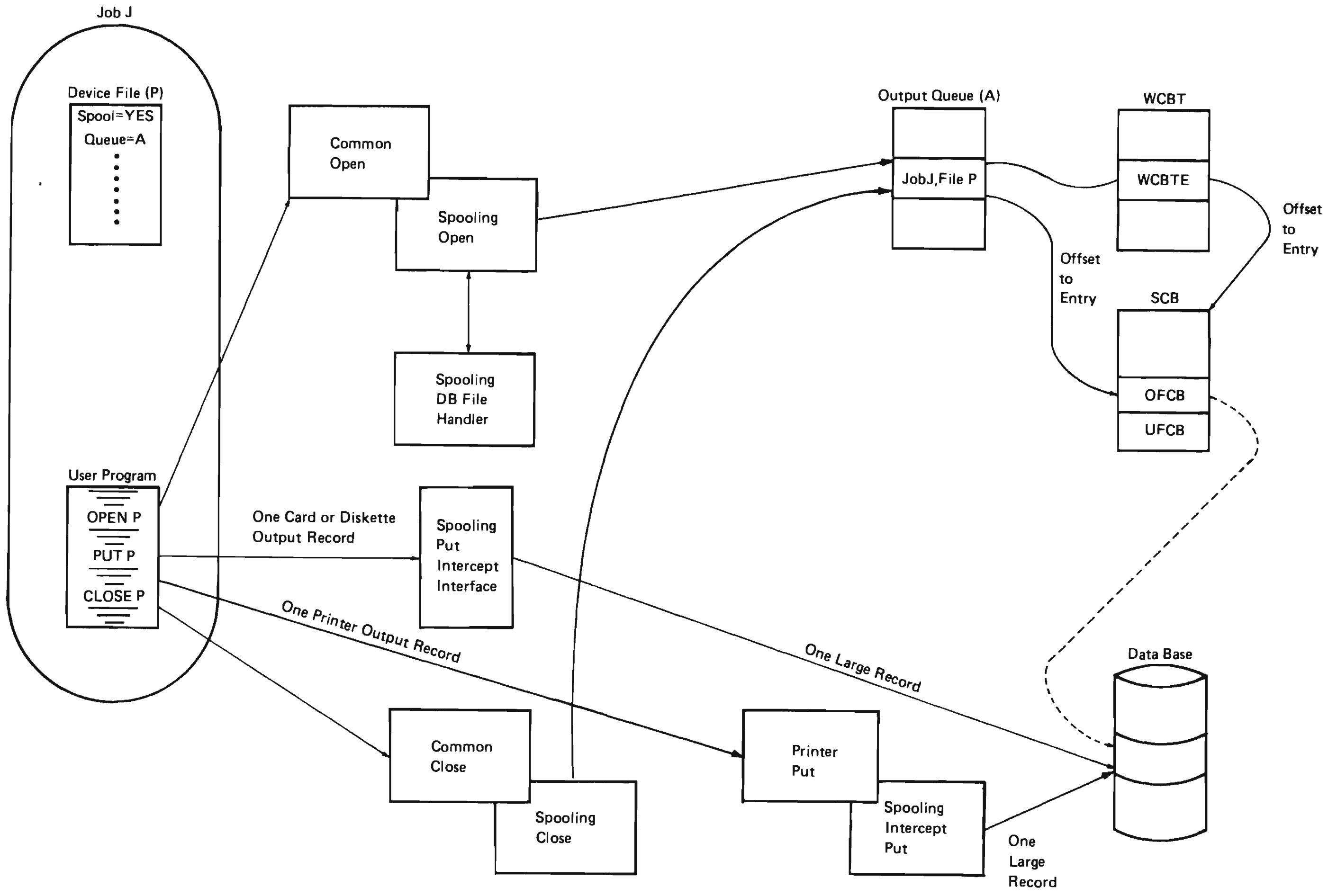


Figure SP-5. Executing a Program that Produces Spooled Output

Writer Producing Spooled Output

Figure SP-6 and the following text describe the operation of a writer that produces spooled output.

Writer output is initiated by a user entering a start writer command. This command invokes the start writer CPP, which generates an entry on the appropriate job queue (spool subsystem queue). An entry for the specified enqueued writer is put into the RWCB (reader/writer control block) along with necessary writer input parameters (such as writer name, device to use for output, identity of the command submitter, and so forth). The output queue is flagged as active to a writer (that a writer has been started to it) so that there is no chance that another writer can be started to the same output queue. Also, events are then sent to active writers for hold or cancel files.

The spool subsystem monitor then requests a job from its job queue. A temporary job structure is allocated and the user-named writer process is created. The writer process is then started to an identified device and output queue. The output queue is flagged as active, so events are sent to active writers for hold or cancel files. When the writer is set up and ready to start producing output files, it obtains the files in a priority and job sequence. However, if the writer is to start with a specific file on the output queue, it will process that file first. The writer then selects the first available file on the queue, processing it and all remaining files in a priority and job sequence. In order for a file to be made available to a writer, the associated queue, job, and the file itself, must not be in a held state. When the writer has obtained the output file, the file is flagged as active to a writer.

Before the selected output file can be sent to the device, the writer must look at the associated OFCB (output file control block) and UFCB (user file control block) of the file, which is within the SCB of the job. The OFCB has information such as the number of copies, the type of form to use, save attribute, the number of file separators, and so forth that is needed by the writer. If the form type is to be changed, the writer issues a message to the system operator. When the forms have been changed, output of the file is begun by:

- Opening the device file with the new output file attributes (done by copying in the attributes from the UFCB of the file's associated SCB)
- Opening the data base file or member that contains the intercepted records

When the preceding has been done, the spooling writer goes into a loop doing data base gets and device function manager puts so that the actual intercepted records can be produced. If the intercepted output goes to the device to which the program intended it to go, the spooling writer does the following:

- If the output is to go to a printer, an entire large record is given to the printer function manager. (If the large records are only 512 bytes long, then eight of them will be combined before they are given to the printer function manager.)
- If the output is to go to the MFCU, the large record is broken up and single puts are done to the MFCU function manager.
- If the output is to go to a diskette, the large record is broken up and single puts are done to the diskette function manager.

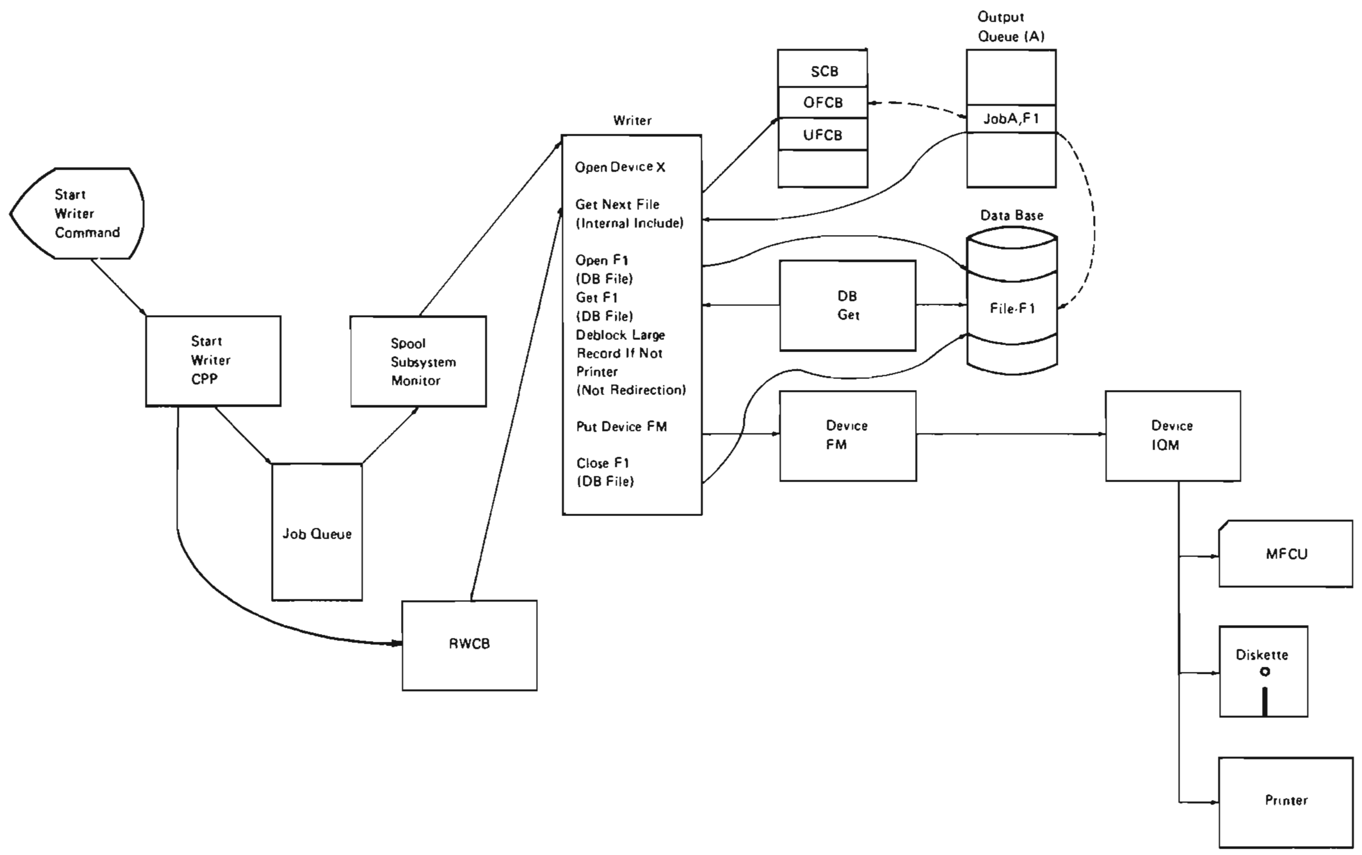


Figure SP-6. Overview of a Spooled Writer Producing Output

Writer Redirection

If the intercepted output from the program is directed to a writer started to a device other than the device originally intended, the writer must perform a redirection. The writer does the following for each type of output:

Intended Output Device	Actual Output Device	Results
Printer	MFCU	Undo the SCS (standard character set); expand back to the original full record and do single puts to the MFCU function manager.
Printer	Diskette	Undo the SCS; expand back to the original full record and do single puts to the diskette function manager.
Printer	Printer	Undo the SCS; expand back to the original full record preceded by a 4-character control code and do single puts to the printer function manager, passing both data and control codes.
MFCU	Printer	Work through the large record and do single puts to a printer function manager. If there is any control information in the large record, it is not used with the printer function manager puts. (SCS is not sent to the printer function manager.)
MFCU	Diskette	Work through the large record and do single puts to the diskette function manager. Control information is not passed to the diskette function manager.
Diskette	Printer	Work through the large record and do single puts to the printer function manager.
Diskette	MFCU	Work through the large record and do single puts to the MFCU function manager.

Notes:
1. When redirection is performed by the writer, the output that goes to the device does not carry control information with it except for printer-to-printer redirection. The control information is stripped off the record before the put to the actual output device function manager. The print function manager receives SCS input from the writer only when the program output and the device are both PRINTER.
2. Printer-to-printer redirection is only done when the printer used by the writer does not support all functions (LPI, CPI, form size) as the intended printer.

This page is intentionally left blank.

Interrelationship of Spooling Control Blocks

Figure SP-7 and the following text describe how the parts of a job fit together with the various spooling-related control blocks and job structures.

The job queues contain entries for batch jobs that are waiting to execute. They are arranged by priority and date, and by time stamp. Every job, batch active, has a WCBTE (work control block table entry) associated with it. The WCBTE contains system information and status of the job throughout the life of the job in the system. It also contains pointers to the SCB (spooling control block) and JMQs (job messages queues) of that particular job. The SCB contains information needed by spooling. That information includes:

- The number of spooled files in the job.
- Status information.
- Information for each inline data file.
- Each spooled output file has associated with it in the SCB:
  - An OFCB (output file control block) that contains file information–file name, file status, data base file member name, and so forth.
  - A UFCB (user file control block) that is a copy of the variable portion of the device file UFCB used by the program at open time so that the writer can produce the file as intended.

Each time that a device file is opened for output and the SPOOL=YES parameter has been specified, another spooled output file is created (unless the previous close was a temporary close) and is put into the SCB of the job. When the file is canceled or the writer is finished with the file, the data base file member is reclaimed for reuse and the entry for the file in the SCB is marked as *this file is done, member is reclaimed.*

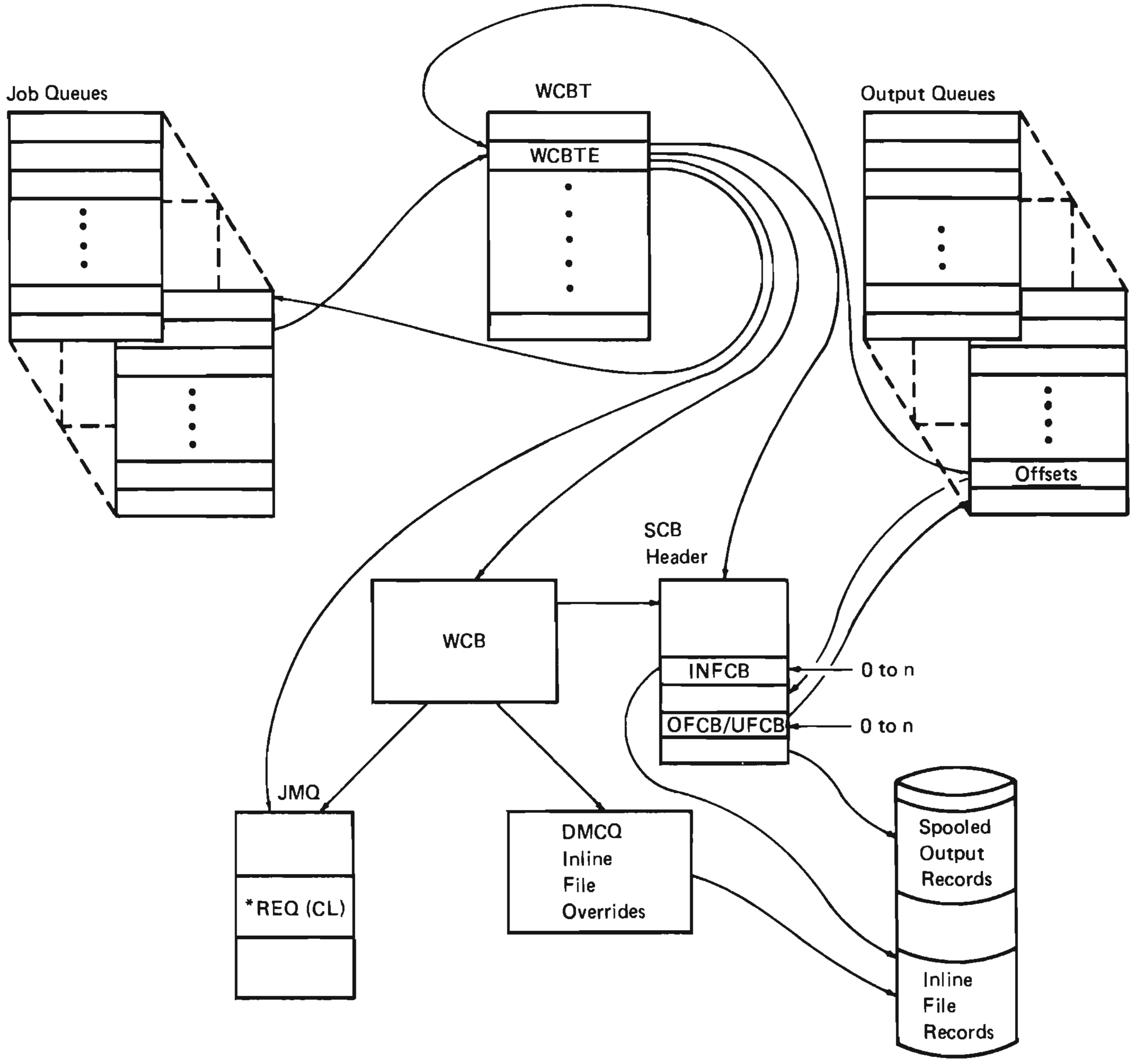


Figure SP-7. Interrelationship of Spooling Control Blocks

The user can have up to 9 998 spooled output files plus one file for the joblog making 9 999 total output files for a job. Each output file has an entry placed on an output queue when the file is opened. The entry that resides in an output queue contains an offset into the WCBTE of the job and the offset into the SCB of the job where the OFCB of the file can be found. Thus, when a queue entry is found, the job can quickly be found in the WCBT and the OFCB of the file can quickly be found in the SCB.

Large Record

When an executing program has its card or diskette output records spooled, the spooling intercept routines accept the put records and block them into a large record. Spooled printer output records are blocked into the large record by a printer (PN) module, which gives control to QSPBPPRT, which in turn puts the large record to the data base file member or updates the file header information. This improves spooling performance because there will be fewer puts to the data base intercept file and fewer gets at write time to obtain the file's large records. For all spooled output, printer, diskette, and MFCU, the output is packed into a large record. Figure SP-8 shows the layout of a large record. It consists of two parts, a header and output data.

Output Data	Header

Figure SP-8. Large Record Layout

Header

Figure SP-9 shows the layout for the header portion of the large record. The header contains all of the required information for the large record.

1 This number is the number of used/valid bytes within this large record. This number reflects the number of bytes that the writer and printer function manager will process for this large record.

2 This represents the attribute bits for advanced print function files. These bits indicate the presence of a page profile and LAC (load alternate character) commands.

3 This number is the number of pages or records contained in this large record. If the output is to be printed, the number is the number of pages ending within the large record. If the output is to the MFCU or a diskette, it is the number of output records within the large record.

4 This number is the number of pages or records contained in this large record. If the output is to be printed, the number is the number of pages beginning within the large record. If the output is to the MFCU or a diskette, it is the number of output records within the large record.

5 This number is the page number or record number of the first page or record within this large record. This is used by the writer so that finding the correct large record is easier and faster.

6 This number is the offset to the page profile if the page profile exists in the large record.

7 This number is the relative record number of the previous large record which contained a page profile.

8 This number is the number of nonblank lines in the first page which ends in the large record.

9 This is the number of nonblank lines in the large record.

10 This number is the offset to the specified page (first, second, third, and so forth).

11 This number is the number of the first line where user data can start on this page relative to the start of the file.

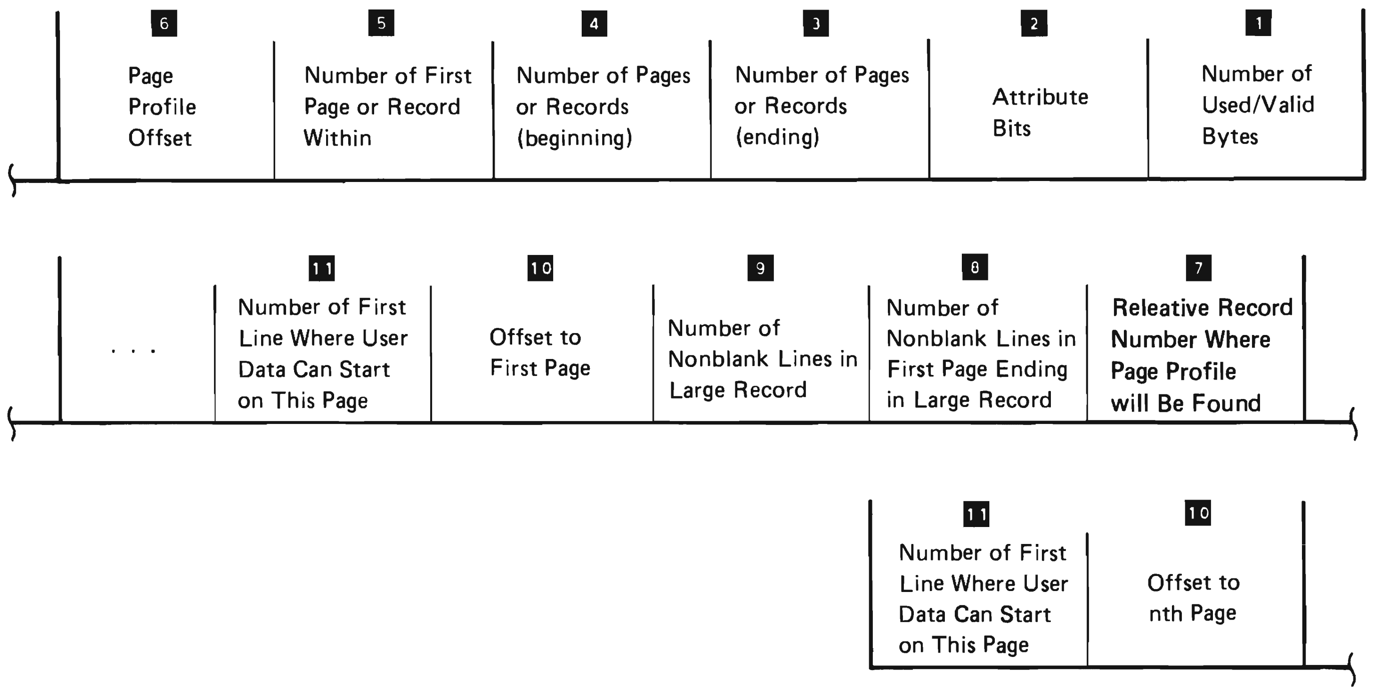

6
Page Profile Offset
5
Number of First Page or Record Within
4
Number of Pages or Records (beginning)
3
Number of Pages or Records (ending)
2
Attribute Bits
1
Number of Used/Valid Bytes
. . .
11
Number of First Line Where User Data Can Start on This Page
10
Offset to First Page
9
Number of Nonblank Lines in Large Record
8
Number of Nonblank Lines in First Page Ending in Large Record
7
Releative Record Number Where Page Profile will Be Found
11
Number of First Line Where User Data Can Start on This Page
10
Offset to nth Page

Figure SP-9. Large Record Header Layout

Output Data

The output data contains the intercepted output records for the printer, MFCU, or diskette. Output records are in the following formats:

- Printer records are in SCS (standard character set) format.
- MFCU records are all of equal length with punch information padded with hex 40s if necessary.
- Diskette records are all equal length, 4096 bytes or less.

Figure SP-10 shows the layout of the output records for print data, MFCU data, and diskette data.

Note: Print output–The print record would be converted to SCS before going into the large record. All printer control language is in SCS; some trailing blanks are eliminated.

MFCU output–At file open time it is determined if a control list will accompany the puts. If it does, every output record will have a 12-byte control list; if not there will not be any control list in the large record. From put to put, the user can vary: punch, no punch with print, no print. Therefore, once spooling open has determined print record length (96 or 128), the record is set in the large record as maximum and padding with hex 40s is done to ensure accurate writer output to the MFCU function manager.

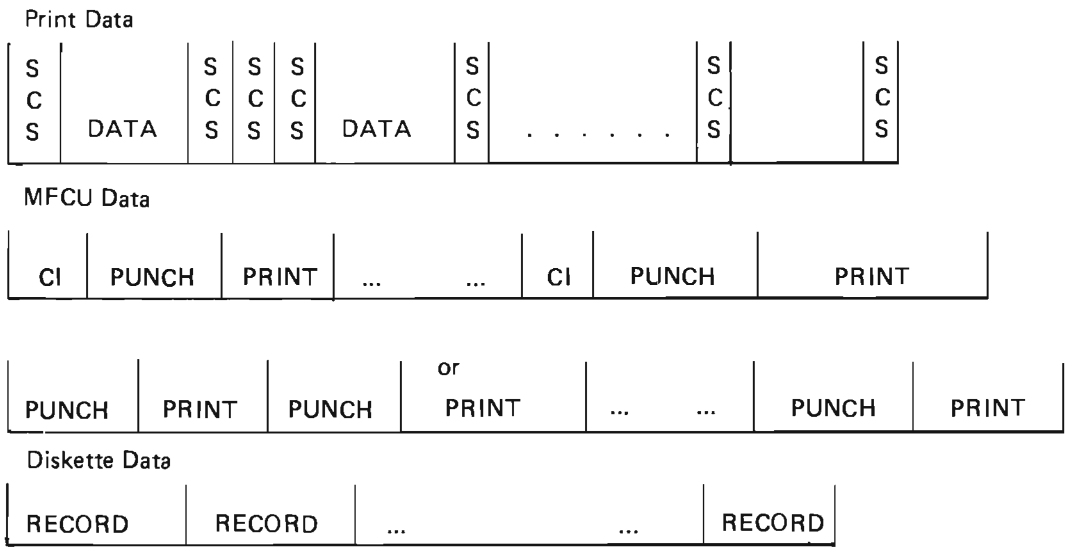


Figure SP-10. Device Output Record Format Layout

INTRODUCTION

The save/restore component of CPF (control program facility) provides the functions needed to save objects offline and later restore those objects to the system. These functions can be used to back up the system, save seldom-used objects to free space for other objects, and to store critical objects offline to prevent unauthorized access to them. Two functions are provided to do this:

- Save objects by writing a copy of the objects to offline storage, or online storage, and optionally free the space occupied by those objects for use by other objects
- Restore saved objects to the system

The use of these functions allows backup copies of entire libraries or individual objects. They also allow a system to recover from failures quickly and easily.

Save Function

The save function writes a copy of an object onto tape or diskettes or to a save file. The object is not removed from the system; it still exists on the system and is available for normal system use. A copy of 1 to 50 libraries, all user libraries, a single object, or a group of objects in one library can be saved in one operation; a single changed object, or a group of changed objects in one library can also be saved in one operation.

A save/restore history about each object saved is maintained by CPF. This information tells when and where each object was last saved and when it was last restored. The information is from the most recent save/restore operation; this lets CPF check that objects restored are the latest copy and not an outdated copy of the object.

The storage space for file, program, and journal receiver objects can also be freed when it is saved. After an object's space is freed, the object is considered to be offline. When an object is offline, its description and offline location are still maintained in the system. However, space from the contents of the object is freed and can be used by other objects.

Freeing an object's space is not the same as deleting an object. When an object is deleted, all information about that object is also deleted from the system. That object must be created or restored to be used again.

Restore Function

The restore function of save/restore copies saved objects back into the system. These restore functions are used to restore any saved objects except certain critical objects in the system library, which are copied back into the system by the installation component.

GENERAL OVERVIEW

Save/Restore Modules

The save/restore component consists of the following modules:

Note: A module identified by an arrow (-->) is an entry module into the component. Indentation of a module shows its dependency on a previous module.

Save Modules

-->QSRPSCPR–Save System (SAVSYS)[1]: This module performs the save system functions as specified by the Save System command. It controls the generation and writing out of the save/restore files included in a system save.

> QSRSVIPL–Save Initial Installation: This module resolves to an object specified in the parameter list, builds an SSR (source/sink request) entry for that object, and then calls QSRSVRQI to issue a machine REQIO to save that object.

[1]This module is a CPP (command processing program).

QSRSVIOC–Save I/O Configuration: This module is the nonstandard object handler for an I/O object (line description, device description, and control unit description).

QSRSVUPR–Save User Profile: This module is the nonstandard object handler for a user profile.

QSYSVAUT–Save Authorized User Table: This module is the nonstandard object handler that extracts the user name and password entries from the authorized user table and saves them.

-->QSRSOCPR–Save Object/Save Changed Object (SAVOBJ/SAVCHGOBJ)[1]: This module interfaces with the command analyzer to perform the Save Object command or the Save Changed Object command. It sets up the save/restore control block to communicate input parameters and environment information between the save/restore modules.

QSRSOCP2–Save Object Processing: This module performs the save object processing. It checks the parameters for validity, interfaces with the librarian for objects to be saved, calls QSRSVCHK, and issues completion messages.

-->QSRSLCPR–Save Library (SAVLIB)[1]: This module performs the save library functions as specified by the Save Library command. It sets up the save/restore control block to communicate input parameters and environment information between the save/restore modules.

QSRSLCP2–Save Library Processing: This module performs the save library processing. It performs all library level processing, including building save/restore descriptors, retrieving a list of objects in the library, and performing OIR (object information repository) maintenance.

The following modules are used by QSRSOCPR and QSRSLCPR:

QSRSVCHK–Object Authority Checks for Save: This module checks that the user has sufficient authority to save each object.

QSRFROBJ–Free Object: This module determines which objects from a list of saved objects are valid to be freed and frees the storage for those objects. (Data base files are freed by QDBSVPST and journal receivers are freed by QJOSAVRC.)

QSRADOPT–Adopt User Profile: This module adopts the QSYS user profile to allow any modules under it to execute with that user profile. This module is also used by QSRRLCPR and QSRROCPR.

The following modules are used by QSRPSCPR, QSRSOCPR, and QSRSLCPR:

QSRSVPRE–Save Pre-I/O module: This module creates a space that will contain a list of all of the objects to save. It locks the objects to be saved if the system is not in restricted state. It interfaces with object handling modules for processing of the pieces of composite objects. It also creates the operand (a source/sink request) that causes the machine REQIO or REQPO function to dump the specified objects.

QDBSVPRE–Save Data Base File Pre-Dump: This module performs the composite object handling needed prior to saving a data base file. It is called once by save/restore for each library that contains data base files.

QDBSVPST–Save Data Base File Post-Dump: This module performs the composite object handling needed for data base files after the data base files have been placed on the save/restore media.

QJOSAVJN–Save Journal: This module performs the object handler function necessary to save a journal.

QJOSAVRC–Save Journal Receiver: This module performs the object handler function necessary to save a journal receiver.

[1]This module is a CPP (command processing program).

QMHSAVMQ–Save Message Queue: This module performs the object handler function necessary to save the descriptions of user-defined message queues.

QSPSAVQ–Save Job/Output Queue: This module performs the function necessary to save the descriptions of job queues and output queues.

QDFSAVDF–Save Save File: This module performs the object handler function necessary to save the description of the save file.

QSRSVRQI–Save REQIO: This module reads the VTOC (volume table of contents) for saves to diskette from the specified volume and then constructs a unique name for the file. QSRSVRQI then issues the machine REQIO or REQPO instructions for the save function.

QSRSVDEQ–This module handles various return codes (other than end of volume) which are in the feedback record from the request I/O performed by QSRSVEOV. It also handles opening and closing of files.

QSRSVPST–Save Post-IO module: This module updates the save/restore history of the object in its OIR, invokes object handlers, and unlocks the objects locked in Pre-IO.

QSRSVEOV–This module performs the dequeue of the request, checks for end of volume, and checks for recoverable error media conditions.

Restore Modules

-->QSRROCPR–Restore Object Command Processor (RSTOBJ)[1]: This module is called by the command analyzer to perform the Restore Object command. It sets up the save/restore control block to communicate input parameters and environment information between the save/restore modules.

QSRROCP2–Restore Object Processing: This module performs the restore object processing. It also does validity, lock, and authority checking at the library level, and performs clean-up and completion notification.

-->QSRRLCPR–Restore Library (RSTLIB)[1]: This module performs the restore library functions as specified by the Restore Library command. It sets up the save/restore control block to communicate input parameters and environment information between the save/restore modules.

QSRRLCP2–Restore Library Processing: This module controls restoration of libraries. Processing includes authority checking, locking, retrieving the contents of a library, clean-up, and completion notification.

-->QSRRUCPR–Restore User Profile (RSTUSRPRF)[1]: This module restores user profiles previously saved by a save system command. Processing includes creating an authorization table to be used by QSRRACPR.

-->QSRRACPR–Restore Authorizations (RSTAUT)[1]: This module restores the private authorizations to the user profiles using the authorization table built by QSRRUCPR.

The following modules are used by QSRROCP2 and QSRRLCP2:

QSRRSCUR–Restore Current: This module uses save/restore history information from the OIR to open a file with the most recent version of the saved objects.

QSRRSFIL–Find File to Restore: This module finds the appropriate file to restore. It searches for a file on the volume matching the criteria specified on the command.

QSRRSLIB–Restore Librarian Interface: This module has two functions depending on how it is called. If it is called and passed a null descriptor pointer, it functions as an interface to the librarian. If it is passed a descriptor pointer, QSRRSLIB builds a list object space and processes the descriptors. It also does validity, lock, and authority checking at the object level.

[1]This module is a CPP (command processing program).

The following module is used by QSRRSCUR and QSRRSFIL:

QSRRSOPN–Open for Restore: This module performs the open functions for restore operations.

The following module is used by QSRRSLIB:

QSRRSOBJ–Restore Object Handling Routine: This module performs the actual restore of the objects. It interfaces with object handling modules for processing of the pieces of composite objects. It builds the source sink request that causes the machine REQIO or REQPO function to load the specified objects. It also optionally generates a listing of objects successfully restored and not restored.

The following modules are used by QSRRSOBJ:

QDBRSPRE–Data Base File Preload: This module performs the object handler function necessary prior to restoring a data base file. QDBRSPRE is called once by save/restore for each restored library that contains data base files.

QDBRSPST–Data Base File Post Load: This module performs the object handler function necessary after the data base files are read in from the media and placed on the system.

QJORSTJN–Restore Journal: This module performs the object handler function necessary to restore a journal.

QJORSTRC–Restore Journal Receiver: This module performs the object handler function necessary to restore a journal receiver.

QMHRSTMQ–Save Message Queue Object Handler: This module performs the object handler function necessary to restore the descriptions of user-defined message queues.

QSPRSTQ–Restore Job/Output Queue: This module performs the object handler function necessary to restore the descriptions of job queues and output queues.

QDFRSTDF–Restore Save File: This module performs the object handler function necessary to restore the descriptions of save files.

The following module is used by QSRRSOPN and QSRRSOBJ:

-->QSRRSRQI–Restore REQIO: This module issues the machine REQIO or REQPO instruction for restore operations. It performs end-of-volume processing and closes the file when the I/O operation is complete.

Display Modules

QSRDSVOL–Display Volume: This module is invoked as the result of a Display Diskette (DSPDKT) command requesting information about objects contained in a save/restore diskette file. Information is displayed for volume, file, or objects within a file.

QSRDSPY–Display Tape: This module is invoked as the result of a Display Tape (DSPTAP) command requesting information about objects contained in a save/restore tape file. Information is displayed for the file or for the objects within the file.

QSRDSAVF–Display Save File: This module is invoked as the result of a Display Save File (DSPSAVF) command. Information is displayed for the file and for objects within the file.

QSRTUNE–Save/Restore Performance Tuning: This module is called by the tape, diskette, and save file function managers when a media is opened for a save, restore, or display operation. It obtains the execution priority of the save/restore job and determines the buffer size to be used by the machine.

Save Commands Overview

Figure SR-1 and the following text describe the functions of the save commands in the save/restore component.

Save System Command

1 The Save System (SAVSYS) command calls QSRPSCPR to save the system.

- A QSRPSCPR calls QSRSVIPL to save the AIPL (alternative initial program load) source and the install program.
- F QSRSVRQI is called to write the library LIB–ALL save/restore descriptor.

D QSRSVPRE is called to save the simple objects. The simple objects to be saved are determined from a list obtained from the librarian component.

B QSRSVIOC is called by QSRSVPRE to perform additional processing of I/O configuration objects.

D QSRSVPRE is called to save the complex objects as defined in a list obtained from the librarian.

C QSRSVUPR and QSYSVAUT are called by QSRSVPRE to perform additional processing of each user profile and user-profile authorizations.

Save Object Command/Save Changed Object Command

2 Save Object/Save Changed Object (SAVOBJ/SAVCHGOBJ) command transfers control to QSRSOCPR. If the process user profile has *save system* special authorization, QSRADOPT G is called to adopt the QSYS user profile. QSRADOPT then calls QSRSOCP2.

4 QSRSOCP2 is called directly by QSRSOCPR if the process user profile does not have *save system* special authorization.

D QSRSOCP2, in turn, calls QSRSVCHK to process the save object request. Information about the objects to be saved is obtained from the librarian.

F QSRSVCHK calls QSRSVPRE to build the save/restore descriptors and REQIO SSR. QSRSOCP2 calls QSRSVRQI to issue the REQIO, QSRSVDEQ to handle return codes and file open/close, QSRSVPST to update S/R history, and QSRSVEOV to handle end of volume and dequeue the REQIO.

K QSRSVDEQ calls QSRSVEOV to handle end of volume and dequeue the REQIO or REQPO.

Save Library Command

3 Save Library (SAVLIB) command transfers control to QSRSLCPR. If the process control user profile has *save system* special authorization, QSRADOPT is called to adopt the QSYS user profile. QSRADOPT then calls QSRSLCP2.

5 QSRSLCP2 is called directly by QSRSLCPR if the process user profile does not have *save system* special authorization.

D QSRSLCP2 calls the librarian to get a list space and then calls QSRSVCHK.

F QSRSVCHK calls QSRSVPRE to build the save/restore descriptors and REQIO SSR. QSRSLCP2 calls QSRSVRQI to issue the REQIO or REQPO.

Save Data Base Files

E QSRSVPRE calls QDBSVPRE and QDBSVPST to save data base files.

Save Journals and Journal Receivers

I QSRSVPRE calls QJOSAVJN for each journal found in the list obtained from the librarian component.

J QSRSVPRE calls QJOSAVRC for each journal receiver found in the list obtained from the librarian component.

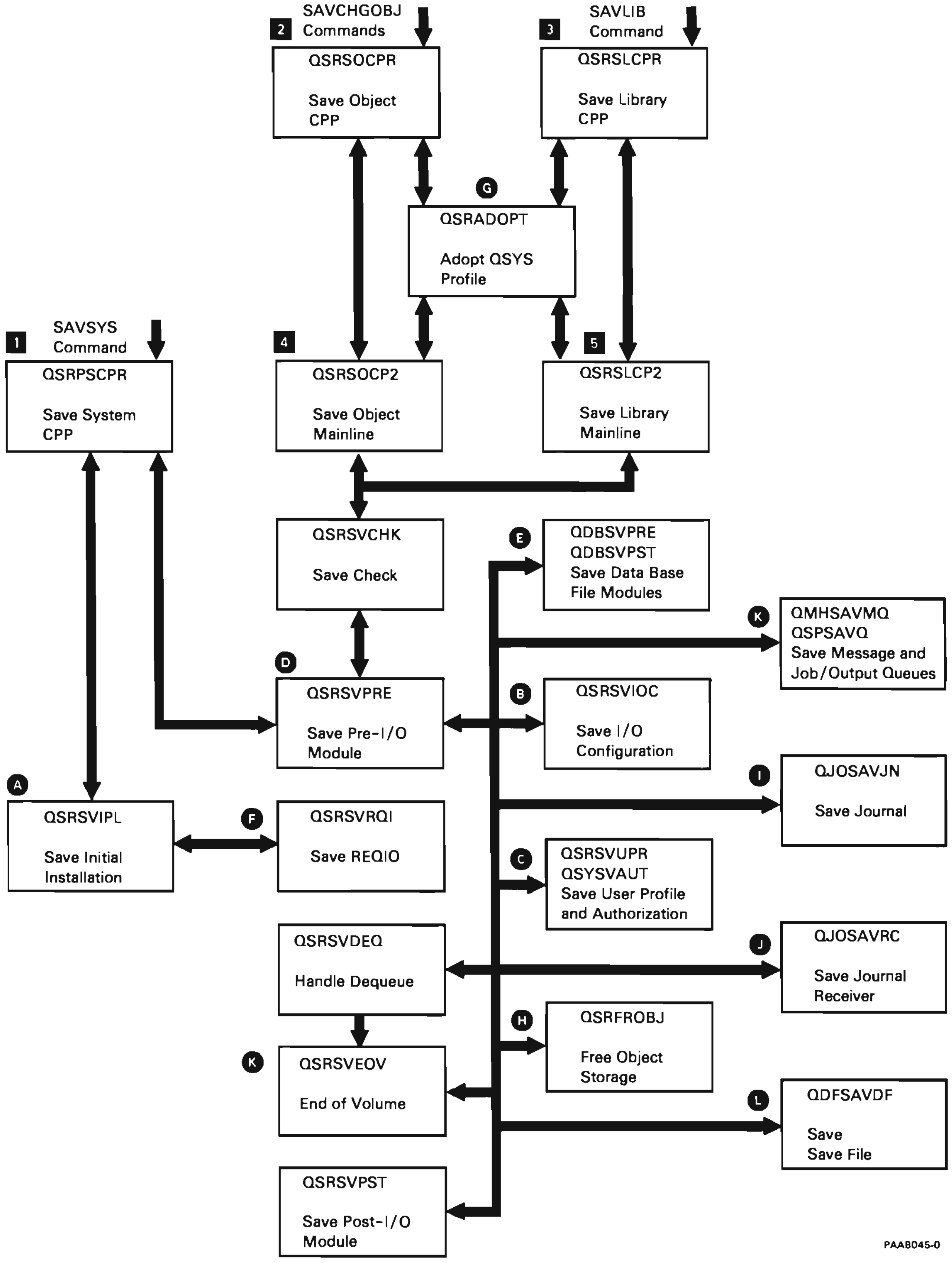
Save Job/Output Queue and Message Queue Descriptions

K QSRSVPRE calls QMHSAVMQ for each user-defined message queue found in the list obtained from the librarian component. QSRSVPRE calls QSPSAVQ for each job queue or output queue found in the list obtained from the librarian component.

L QSRSVPRE calls QDFSAVDF for each save file found in the list obtained from the librarian component.

Free Object Storage

H QSRSVPRE calls QSRFROBJ if storage is to be freed for the saved object.



SAVCHGOBJ Commands
2
QSRSOCPR Save Object CPP
SAVLIB Command
3
QSRSLCPR Save Library CPP
G
QSRADOPT Adopt QSYS Profile
SAVSYS Command
1
QSRPSCPR Save System CPP
4
QSRSOCP2 Save Object Mainline
5
QSRSLCP2 Save Library Mainline
QSRSVCHK Save Check
E
QDBSVPRE QDBSVPST Save Data Base File Modules
K
QMHSAVMQ QSPSAVQ Save Message and Job/Output Queues
D
QSRSVPRE Save Pre-I/O Module
B
QSRSVIOC Save I/O Configuration
A
QSRSVIPL Save Initial Installation
F
QSRSVRQI Save REQIO
I
QJOSAVJN Save Journal
C
QSRSVUPR QSYSVAUT Save User Profile and Authorization
QSRSVDEQ Handle Dequeue
J
QJOSAVRC Save Journal Receiver
H
QSRFROBJ Free Object Storage
K
QSRSVEOV End of Volume
L
QDFSAVDF Save Save File
QSRSVPST Save Post-I/O Module
PAAB045-0

Figure SR-1. Save Commands Overview

This page is intentionally left blank.

Restore Commands Overview

Figure SR-2 and the following text describe the restore commands functions.

Restore Library Command

1 The Restore Library (RSTLIB) command transfers control to QSRRLCPR. If the process user profile has *save system* special authorization, QSRADOPT A is called to adopt the QSYS user profile. QSRADOPT then calls QSRRLCP2. QSRRLCP2 is called directly by QSRRLCPR if the process user profile does not have *save system* special authorization.

2 If the SAVLIB parameter is *NONSYS, a list of libraries to be restored must be read. If the VOL parameter is *SAVVOL, QSRRSCUR is called; otherwise QSRRSFIL B is called. Both QSRRSCUR and QSRRSFIL calls QSRRSOPN F to open the files and read the list of libraries. QSRRSOPN calls QSRRSRQI G to read the list of libraries. Control is returned to QSRRLCP2. For each library in the library list, QSRRSFIL B is called to find the library's save/restore file.

If the SAVLIB parameter names a library, a list of libraries does not have to be read. If the VOL parameter is *SAVVOL, QSRRSCUR C is called; otherwise QSRRSFIL is called. Both QSRRSCUR and QSRRSFIL call QSRRSOPN to open the file. Control is returned to QSRRLCP2.

For each library to be restored, QSRRLCP2 calls QSRRSLIB D to perform common processing (that is, find the object in the library, option check, authority check, and locking). QSRRSLIB calls QSRRSOBJ E to perform object-type specific processing and to build the I/O request.

QSRRSOBJ calls QSRRSRQI G to perform the I/O operation and, if necessary, to handle any I/O errors.

Object handlers H are called to perform the object handler functions as necessary.

Restore Object Command

3 The Restore Object (RSTOBJ) command transfers control to QSRROCPR. If the process user profile has *save system* special authorization, QSRADOPT A is called to adopt the QSYS user profile. QSRADOPT then calls QSRROCP2. QSRROCP2 is called directly by QSRROCPR if the process user profile does not have *save system* special authorization.

4 QSRROCP2 calls QSRRSCUR C if the VOL parameter on the Restore Object command is *SAVVOL. If the VOL parameter is not *SAVVOL, QSRRSFIL B is called.

Both QSRRSCUR and QSRRSFIL call QSRRSOPN F to open the file and read the description of the file's objects. QSRRSOPN calls QSRRSRQI G to restore the object descriptions. Control is returned to QSRROCP2.

QSRROCP2 calls QSRRSLIB D to perform the object type not specific processing. QSRRSLIB calls QSRRSOBJ E to perform the object type specific processing and to build the I/O request.

QSRRSOBJ calls QSRRSRQI G to perform the I/O operation and, if necessary, to handle any I/O errors.

Object handlers H are called to perform the object handler functions as necessary.

Restore User Profile

5 The Restore User Profile (RSTUSRPRF) command transfers control to QSRRUCPR.

B QSRRSFIL is called by QSRRUCPR to identify the save/restore file to be used in the restore user profile operation.

F QSRRSOPN is called by QSRRUCPR to open the save/restore file.

G QSRRSRQI is called to perform the I/O operations and, if necessary, to handle any I/O errors.

6 The Restore Authority (RSTAUT) command transfers control to QSRRACPR. QSRRACPR restores the private authorizations to the user profiles using the authorization table built by QSRRUCPR 5.

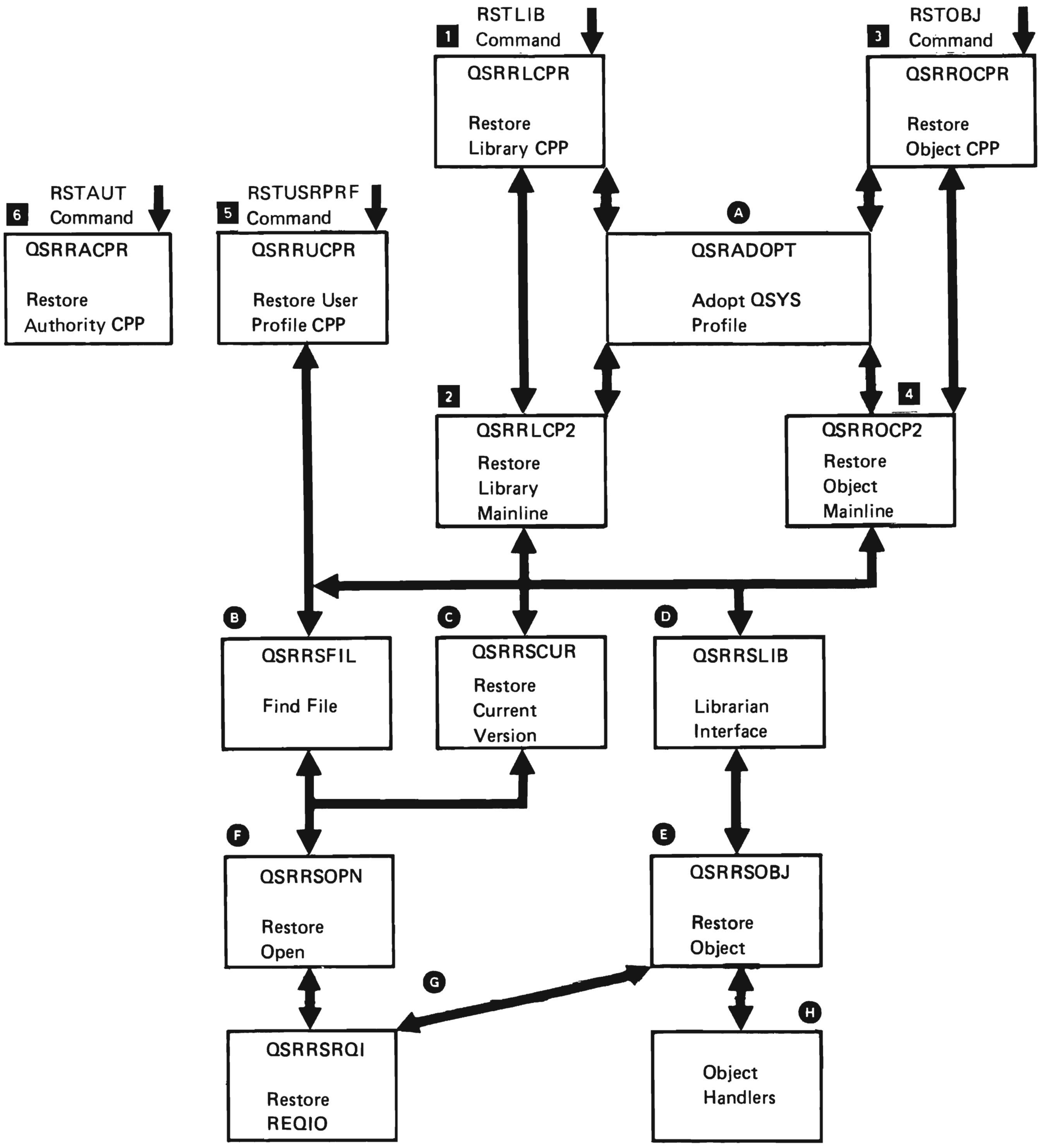


Figure SR-2. Restore Commands Overview

Composite Object Interface

This interface provides two functions: save and restore for both standard composite objects and nonstandard composite objects.

Standard Composite Objects

A standard composite object, for save/restore purposes, has the following characteristics:

- Is structured as a composite object (see Figure SR-3. Standard Composite Object Structure).
- The primary and secondary objects are valid save/restore objects.
- The secondary object cannot be the primary object.
- The object will be restored in exactly the same state that it was at the time of the save and with the same secondary objects in the same order.
- Only the primary object needs to be locked to reliably perform a save, restore, or suspend function.

Standard Composite Objects—Save Function

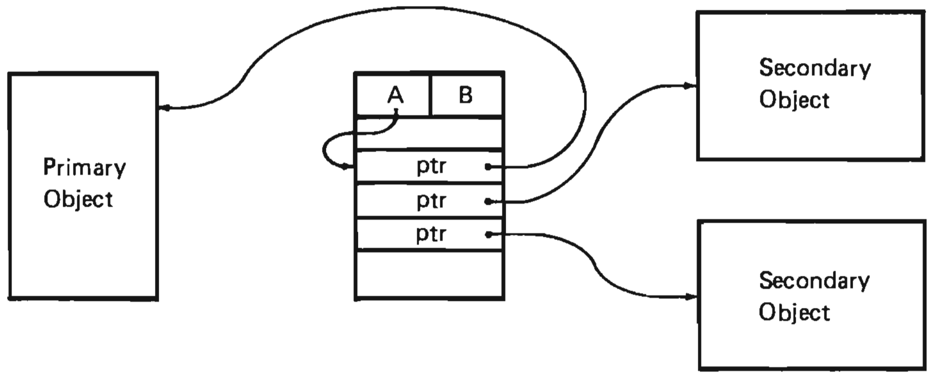
The save function saves the primary object and all secondary objects.

Standard Composite Objects—Restore Function

The restore function restores the primary object, all the secondary objects, and resets the pointers to the primary and secondary objects. If the secondary objects on the system do not match the secondary objects on the save media with respect to name, type, subtype, and position, the objects will not be restored.

Nonstandard Composite Objects

A nonstandard composite object is an object whose internal structure is not known by save/restore. To save or restore it requires an object handler, which is normally supplied by the component responsible for the object.



A – Offset to the pointer list.

B – Number of pointers in the list.

Figure SR-3. Standard Composite Object Structure

Save/Restore Function Manager

Figure SR-4 and the following text describe the structure and the functions of the modules.

1 QSROPEN, Open Online Save File, is responsible for the file specific operations needed to open an online save file. The functions performed are:

- Ensure exclusive file allocation by checking the DMCQ for other open file ODPs to this online save file.
- Allocate and initialize the SR work area and the record buffers in the ODP.
- Validate all file open parameters from the save file, UFCB, and overrides.
- Adjust the ODP Entry Point Table and fill the Common Open Feedback area information, including the current file record count.
- Clear the save file, if necessary, for an output file.
- Establish a load/dump session for the dump space object if the file was opened for use with a save or restore operation.

2 QSRCLOSE, Close Online Save File, is responsible for the file specific operations needed to close an online save file. The functions of this module are:

- Flush output data by writing all buffered records into the dump space.
- Terminate the load/dump session for the dump space object that was established when the file was opened for use with a save or restore operation.
- Modify the dump object to release all possible unused allocated space in the dump space object, and ensure that all records are safely written to non-volatile storage.

3 QSRGET, Get Record, is responsible for retrieving a block of records from the dump space part of the file, optionally deblocking them, and passing the records to the using program. The functions performed by this module are:

- Retrieve a block of records from the dump space object whenever the current buffer is exhausted.
- Deblock the input records and pass them to the using program.

4 QSRPUT, Put Record, is responsible for blocking records received in the output buffer from the user and inserting them into the dump space object part of the file. The functions performed by this module are:

- Receive records from the using program and block them in the output buffer.
- Insert a block of records into the dump space object whenever the current buffer is full.

5 QSRFEOD, Force End of Data is responsible for forcing any buffered output records into the dump space object part of the file, and for signaling end of file for an input file. The functions performed by this module are:

- Signal end of file for an input file.
- Insert any buffered output records into the dump space object part of the file.
- Ensure that all output records are safely written to non-volatile storage.

6 QSRFMERR, Function Manager Error Handler, is invoked to build the replacement text and send any message that must be sent by the Save/Restore function manager.

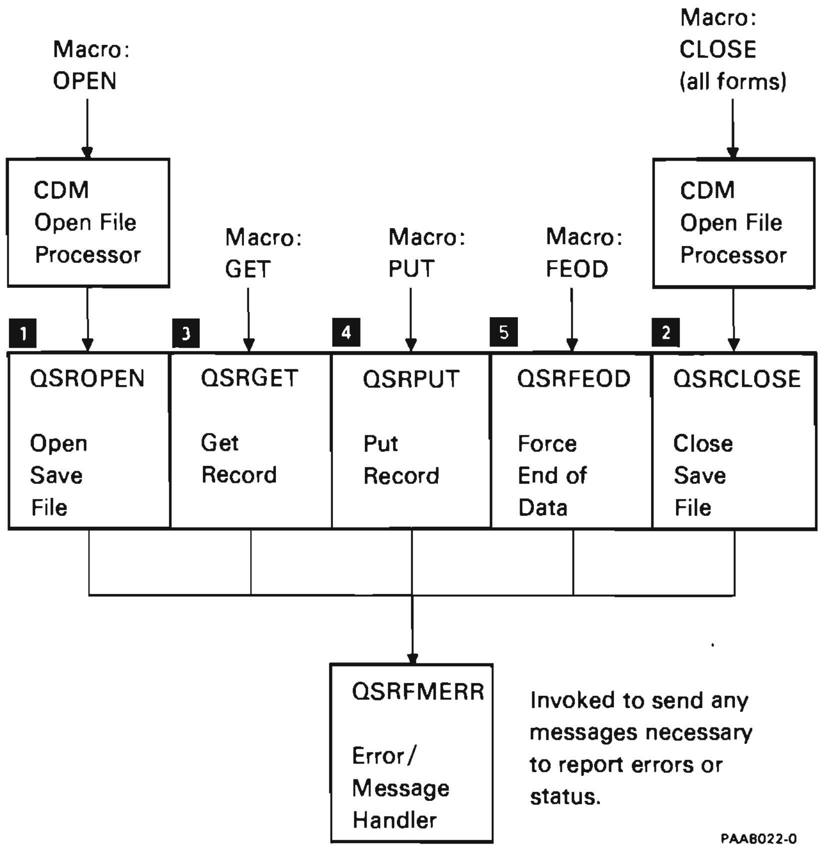


Figure SR-4. SR Function Manager Structure

Switched Lines

INTRODUCTION

The switched lines component of the CPF (control program facility) is composed of two major functions:

- Communications services
- Logical unit services

The communications services consist of a group of event handlers that run under the system arbiter process (QSYSARB), and provide the following functions:

- Switched line connection and disconnection: Establishing manual autoanswer switched line connections, establishing manual and auto-dial switched line connections, disconnecting switched lines, and allowing devices to be obtained prior to successful contact.
- Error recovery: Automatically recovering ND (network description), CD (control unit description), and LUD (logical unit description) errors, and stopping and resuming error recovery.

The logical unit services consist of a group of event handlers that run under the logical unit services process (QLUS), and provide the following functions (for advanced program-to-program communications only):

- Establishment of event monitors at LUD (logical unit description) creation and IPL (initial program load time)
- Vary on LUD and vary off LUD
- Disconnection of switched connections
- Change the maximum number of allowed sessions with the remote system

GENERAL OVERVIEW

Communications Modules

The switched lines component consists of the following communications services modules:

Note: An arrow (-->) identifies a module as being an entry into the component. Indentation of a module shows its dependency on a previous module.

-->QSWALLOC–Switched Line LUD Allocate: This module is called to signal the LUD-allocated event to the system arbiter process.

-->QSWANSWR–Answer Line (ANSLIN)[1]: This module processes the Answer Line command.

-->QSWCDCPP–Stop/Resume Control Unit Recovery (STPCTLRCY/RSMCTLRCY)[1]: This module processes the Stop Control Unit Recovery and Resume Control Unit Recovery commands.

-->QSWCUDEV–CD Event Handler: This module routes and handles the event and calls the CD event processor for all machine interface CD events.

> QSWCDCR–CD Unsuccessful Contact Event Processor: This module processes unsuccessful CD contact events, and sends a message to the system operator message queue.
>
> QSWCDFR–CD Failure Event Processor: This module processes CD failure events, and sends a message to the system operator message queue.
>
> QSWCDLST–CD Lost Contact Event Processor: This module processes lost CD contact events, and sends a message to the system operator message queue.

[1]This module is a CPP (command processing program).

QSWCDNAV–CD Unavailable Event Processor: This module processes CD unavailable events, and sends a message to the system operator message queue.

QSWCDPRV–CD Protocol Violation Event Processor: This module processes CD protocol violation events, and sends a message to the system operator message queue.

QSWCDSUC–CD Successful Contact Event Processor: This module processes successful CD contact events, and sends a message to the system operator message queue.

-->QSWDIAL–Switched Line Dial Out: This module issues a machine interface Modify CD (Dial) instruction to perform a manual or auto-dial operation.

-->QSWDLTLD–Switched Line Delete LUD Entry: This module is used to delete an existing LUD entry in the CD-associated space.

-->QSWHRCMN–Hold/Release Communications Device (HLDCMNDEV/RLSCMNDEV)[1]: This module processes the Hold Communications Device and Release Communications Device commands.

-->QSWINTQ–Initialize Message Queue: This module initializes the message queue.

-->QSWINTSP–Initialize CD-Associated Space: This module initializes the CD-associated space.

-->QSWLDCPP–Stop/Resume Device Recovery (STPDEVRCY/RSMDEVRCY)[1]: This processes the Stop Device Recovery and Resume Device Recovery commands.

-->QSWLOCLD–Locate LUD Entry: This module is used to locate an existing LUD entry in the CD-associated space.

-->QSWLUDEV–LUD Event Handler: This module is used to process successful and unsuccessful LUD contact events, and LUD failure events.

-->QSWNDCPP–Stop/Resume Line Recovery (STPLINRCY/RSMLINRCY)[1]: This module processes the Stop Line Recovery and Resume Line Recovery commands.

-->QSWNDEV–ND Event Handler: This module routes and handles the event and calls the ND event processor for all machine interface ND events.

QSWNDFR–ND Failure Event Processor: This module processes ND failure events, and sends a message to the system operator message queue.

QSWNDNOR–ND Disconnect Event Processor: This module processes ND disconnect events, and sends a message to the system operator message queue.

-->QSWUNUSE–Switched Line Unusable: This module is called to mark a CD and its attached LUD vary on unusable. This is done when an error occurs that requires the CD to be varied off in order to recover.

Logical Unit Services Modules

The switched line component consists of the following logical unit services modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QSWCNSCP–Change Session Maximum (CHGSSNMAX)[1]: This module processes the Change Session Maximum command.

-->QSWIDLES–Switched Line LUD Idle Sessions Event Handler: This module handles the LUD idle session event, and initiates the unbinding of sessions.

-->QSWILUSM–Initial Logical Unit Services Monitor: This module creates the event monitors for the LUDs that exist at start CPF time.

[1]This module is a CPP (command processing program).

-->QSWLCH1–Logical Unit Services LUD Control Event Type 1 Handler: This module handles the LUD control event type 1 (vary on, vary off, and LUD successful contact).

QSWILCS–Logical Unit Services LUD Successful Contact Event Processor: This module processes the successful LUD contact event.

QSWIVOFF–Logical Unit Services LUD Vary Off: This module performs the LUD vary off.

QSWIVON–Logical Unit Services LUD Vary On: This module performs the LUD vary on.

-->QSWLCH2–Logical Unit Services LUD Control Event Type 2 Handler: This module deletes and creates the event monitors for LUD support and control event type 1 handlers.

-->QSWLSH1–Logical Unit Services LUD Support Event Type 1 Handler: This module handles the LUD support event type 1 (change number of sessions request).

QSWCNSC–Logical Unit Services Change Number of Sessions Complete: This module is the REQIO completion program for REQIO change number of sessions.

QSWIOCMP–Logical Unit Services I/O Completion: This module provides routing when the logical unit services REQIOs complete.

QSWSALC–Logical Unit Services Source Program Conversation Allocation: This module allocates a source conversation on the reserved modename to send change number of sessions requests to the target program.

QSWSLCC–Logical Unit Services Program Allocation Complete: This module builds and sends the change number of sessions request after the reserved modename is allocated.

QSWSRCVC–Logical Unit Services Source Program Change Number of Sessions Receive Complete: This module handles the change number of sessions request reply sent by the target program.

-->QSWNOSES–Switched Line CD No Sessions Event Handler: This module handles the CD no sessions event, and calls QSWABAND to disconnect the switch connection.

-->QSWSWERP–Miscellaneous Error Handler: This module is called to handle miscellaneous errors in the logical unit services process, and sends escape and/or information messages to the system operator message queue.

-->QSWTALC–Logical Unit Services Target Program Conversation Allocation: This module receives change number of sessions requests from the source program and sends the reply.

QSWTALCC–Logical Unit Services Target Program Conversation Allocation Complete: This module monitors for successful and unsuccessful completion of REQIOs used to send the change number of sessions request reply.

-->QSWABAND–Disconnect a Switched Line Connection: This module issues a machine interface Modify CD (Abandon) instruction to disconnect an active switched line connection.

QSWCDMAN–CD Manual Intervention Event Handler: This module processes CD manual intervention events, and sends a message to the system operator message queue.

-->QSWDAMGE–Partial Damage Recovery: This module serves as the event handler for the partial system object damage set event. Source/sink objects are handled here and other objects are handled by QRCPDMGL. The source/sink objects will be marked as failed, and a message will be sent to the system operator.

-->QSWCPFEV–CPF Switched Line Event Monitor: This module will route the switched line events and the lock events to the proper handlers. The decisions, about which module is invoked, are based on whether the event can possibly cause the disconnection of a remote switched CD.

QSWDISC–CPF Event Processor: Disconnecting. This module handles some of the requests of the CPF switched line request event and the object locked event. The object locked event and specific switched line requests are handled here because they could possibly cause the disconnection of a switched CD.

QSWNDISC–CPF Event Processor: Non-disconnecting. This module handles all the switched line CPF signaled events that cannot cause the disconnection of a switched CD and the object lock time-out event.

-->QSWREPLY–Inquiry Message Reply Routing Module: This module serves as the break message handling program for the switched line message queue. Any inquiry message replies received in this queue are routed to the appropriate message handler.

QSWCDCRH–CD Contact Unsuccessful Message Reply Handler: This module handles replies to the CD contact unsuccessful messages. This module performs either a machine interface Modify CD (Cancel) or Modify CD (Continue) depending upon the reply.

QSWCDFRH–CD Failure Reply Handler: This module handles replies to inquiry messages sent as the result of a CD failure event. This module performs either a machine interface Modify CD (Cancel) or Modify CD (Continue) depending on the reply.

QSWCDMRH–CD Miscellaneous Message Reply Handler: This module handles replies to messages not covered by another reply handler.

QSWLUDRH–LUD Bind Queued Reply Handler: This module handles replies to the LUD bind queued inquiry message. This module may perform a machine interface Modify LUD (Cancel) depending on the reply.

QSWNDFRH–ND Failure Reply Handler: This module handles replies to ND failure messages. This module performs either a machine interface Modify ND (Cancel) or a Modify ND (Continue) depending upon the reply.

-->QSWSNDMS–Send Message Module: This module will determine and send the appropriate message to the system operator or history log. The appropriate message depends on whether the object is an ND or a CD and what the error code is.

Communications Services Events Signaled by Other CPF Components

The information communicated by communications services events signaled by other CPF components to the switched line component include:

Events Handled by QSWCPFEV

- Obtain device
- Obtain device response
- Switched request

Communications Services Events Signaled by the Machine

The information communicated by communications services events signaled by the machine to the switched line component include:

Events Handled by QSWCPFEV

- Object lock request granted
- Asynchronous lock time-out

Events Handled by QSWCUDEV

- CD contact successful
- CD contact unsuccessful
- CD lost contact
- CD failure
- CD protocol violation
- CD unavailable
- CD manual dial required
- CD switched connection required
- CD leased contact required

Event Handled by QSWDAMGE

- Object partial damage

Events Handled by QSWLUDEV

- LUD contact successful
- LUD contact unsuccessful

Events Handled by QSWNDEV

- ND synchronous data link control connection failure
- ND binary synchronous communications connection failure
- ND disconnection failure
- ND multi-leaving telecommunications access method connection failure
- ND peer device connection failure
- ND failure
- ND protocol violation

Logical Unit Services Events Signaled by Other CPF Components

The information communicated by logical unit services events signaled by other CPF components to the switched line component include:

Event Handled by QSWCNSCP

- Change Session Maximum command completion

Events Handled by QSWLCH1

- Vary on
- Vary off
- Contact successful

Events Handled by QSWLCH2

- Peer device creation
- Peer device deletion

Events Handled by QSWLSH1

- Change Session Maximum command issued
- Peer device negotiation required

Event Handled by QSWTALC

- Peer device pass conversation

Logical Unit Services Events Signaled by the Machine

The information communicated by logical unit services events signaled by the machine to the switched line component include:

Event Handled by QSWIDLES

- Peer device idle session

Event Handled by QSWNOSES

- Peer device no sessions

Communications Overview

Figure SW-1 shows an overview of the communications services.

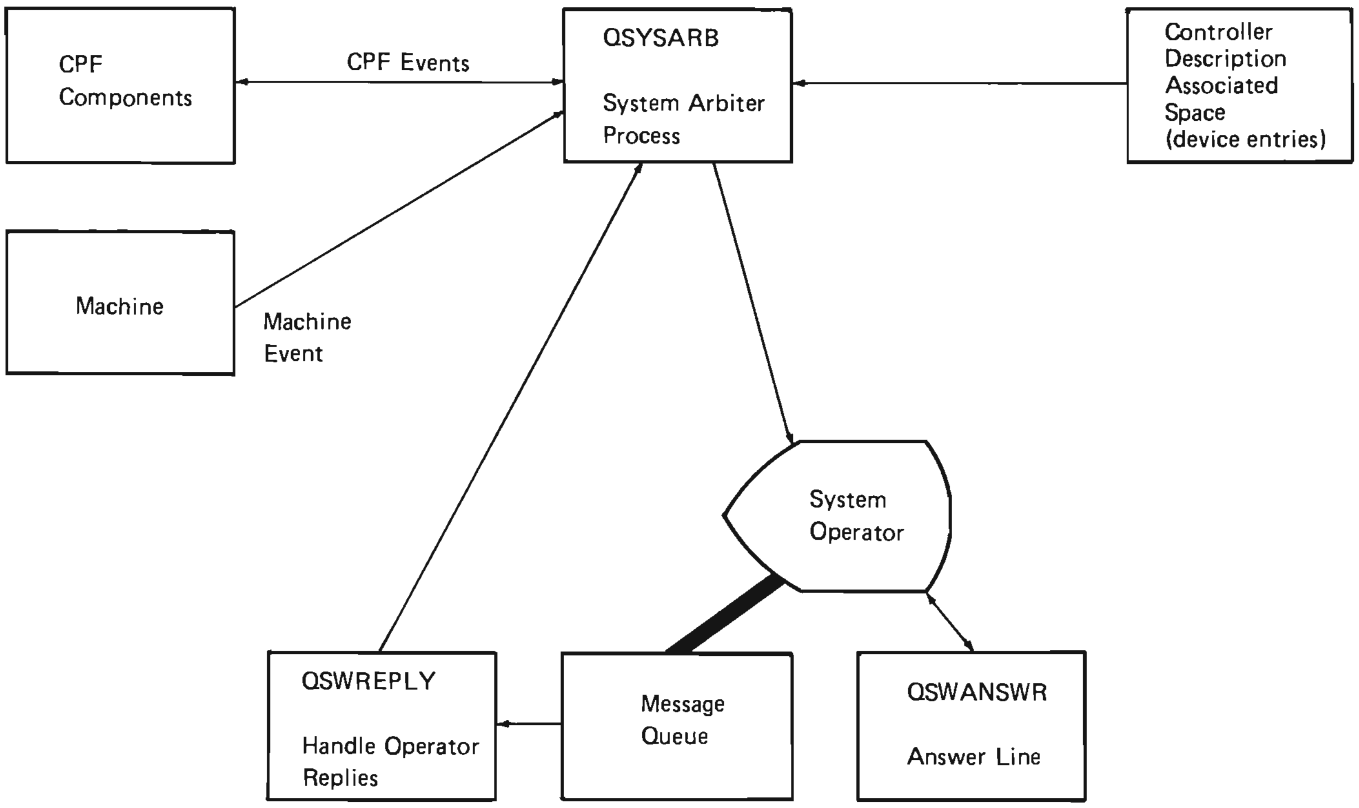


Figure SW-1. Communications Overview

Logical Unit Services Overview

Figure SW-2 shows and overview of the logical unit services.

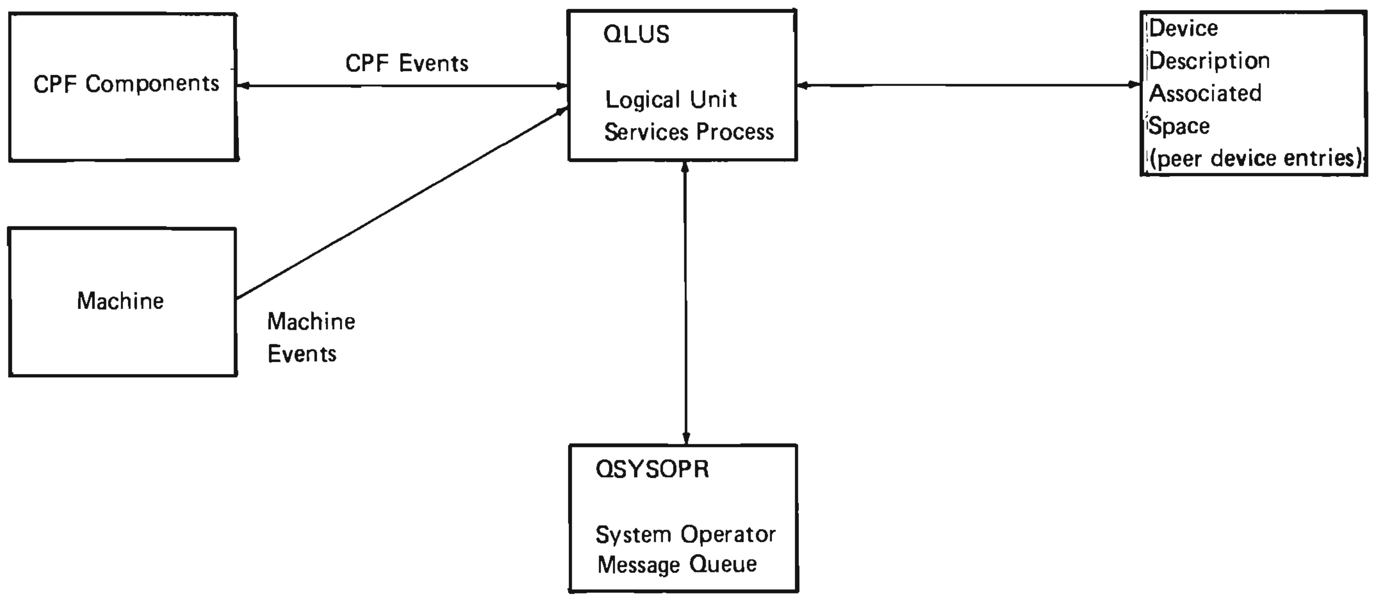


Figure SW-2. Logical Unit Services Overview

INTRODUCTION

The security component of the CPF (control program facility) provides the controls in System/38 that ensure data integrity and data security. Data integrity is the protection of programs and data from accidental alteration or destruction. Security is the prevention of access to and use of programs and data by unauthorized users. Directly related to integrity and security is the need for user identification. User identification ensures that the programs and data which the user is authorized to use are made available to the user by the system. The security component provides user profiles as the means of user identification and also provides the means for authorizing user access to specific objects.

User Profile

The user profile is an object that identifies and represents a specific user to the CPF. The user profile is a collection point for all of the security information related to a user. If a new user wants to use the system, a user profile must be created or made available for that user to use. The user profile contains:

- User name: This is the name by which the user is known to the system. Each name must be unique. Authorizations are made to the user name.

- Owned objects: This is a list of all of the objects that the user owns.

- Authorized objects: This is a list of the objects that the user is authorized to use and includes the rights of use authorized for those objects.

- Authorized users: This a list of the users authorized to use the objects owned by this profile and includes the rights of use explicitly authorized.

- Attributes: This is a list of the special attributes which the user is authorized to use. The attributes of a user profile determine which special authorizations are assigned to that user. Special authorizations are required by the system to execute certain functions. The two attributes that can be authorized to a user are:
  - Save system attribute: The save system attribute authorizes the user to save, restore, and free space for any object on the system.
  - Job control attribute: The job control attribute is the authority needed to change, cancel, display, hold, and release jobs other than the user's own. It also lets the user change, clear, display, hold, and release all spooling queues that allow operator control.

- Storage: This is the maximum amount (in K-bytes) of auxiliary storage that can be allocated for the storage of permanent objects owned by the user and the amount of storage currently being used.

- Privileged instruction authorization: The security officer is the only user profile authorized to use the privileged instructions needed to create and change user profiles. All user profiles are authorized to use the privileged instructions needed to create devices (device descriptions, control unit descriptions, and line descriptions).

Note: The security officer is the only one able to delete user profiles and to display the list of authorized users, even though these operations are not tied to privileged instructions. He is the only one authorized to access the AUT (authorized user table).

User Profile Associated Space

Each user profile has an associated space containing the following information:

- Initial Program: This is the name of the initial program to be invoked whenever the user signs on the system.
- Priority Limit: This is the priority limit authorized to the user.
- Job Description: The name of the job description to be used to set the attributes for the user's job.
- Accounting Code: This is the accounting code to be used to set the attributes for the user's job.
- Message Queue: The name of the message queue associated with the user.
- Output Queue: The name of the output queue associated with the user.

User Password

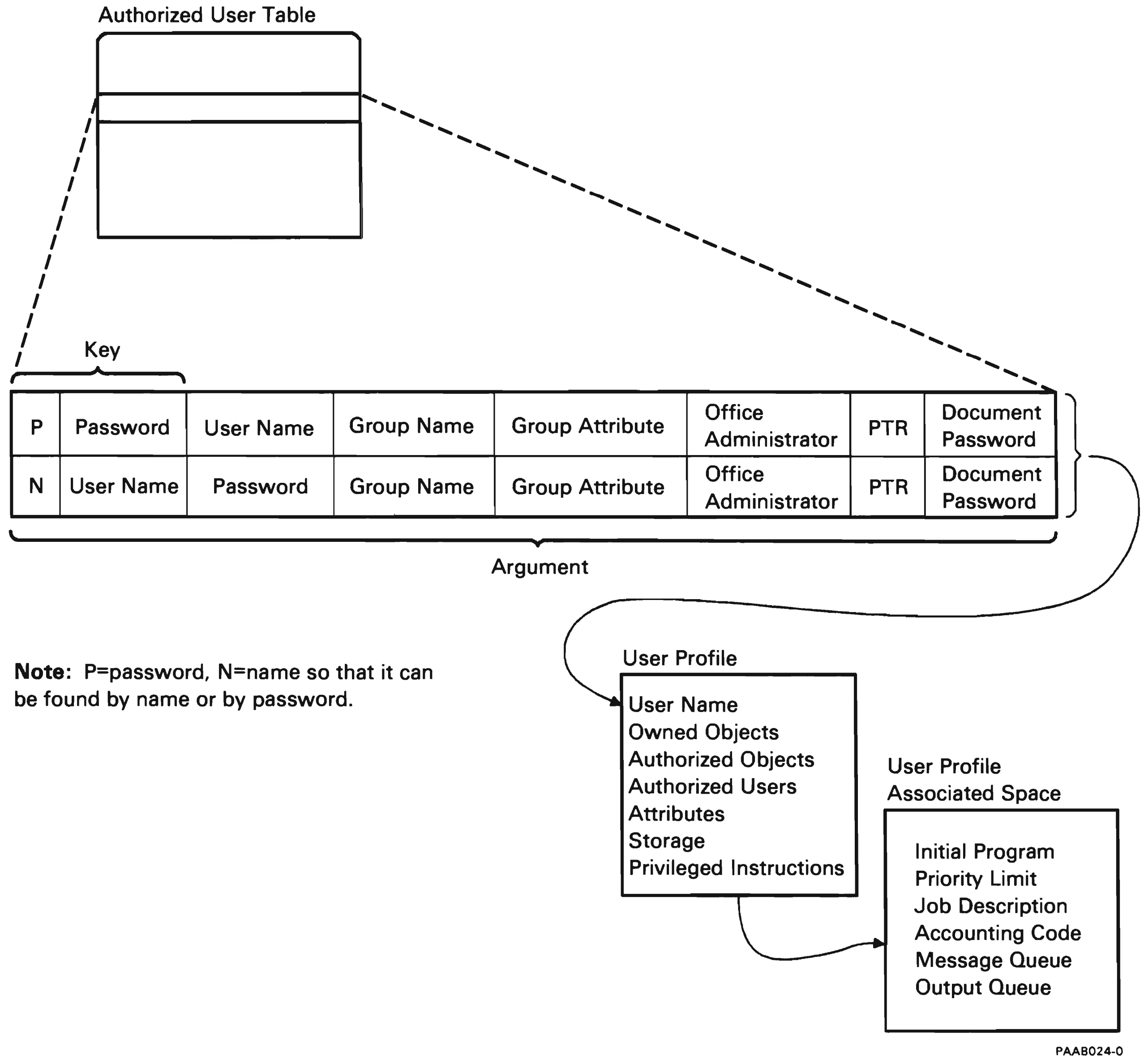
Access to a user profile and its use is controlled by the user password. The CPF uses a user password to determine which user profile represents that particular user. A user password should be known to only the people who use it. To prevent unauthorized use of a password, the security officer can change it periodically.

Authorized User Table

User passwords are maintained in a system object (independent index) called the AUT (authorized user table). Addressability to the AUT is maintained in QSYS and the name of the AUT is QSYUPTBL. It is a cross reference list of user passwords to user profiles. The AUT contains the following information about each user:

- User password: This is the key to the user profile by which the user is identified to the system. Each password must be unique and each can only be associated with one user profile.
- User name: This is the name of the user profile for which this password is the key.
- User profile address: This is a pointer to the user profile for which this password is the key.
- Group profile: The name of the user profile to be used in conjunction with this user profile to determine a job's authority to an object.
- Group attribute: An indicator of what authority is to be granted to the group profile when this user profile creates an object.
- Office administrator definition: This is a special right given to the user who is defined to be the administrator for the OFFICE/38–Personal Services/38 product.
- Document Password: This is the password that allows an affinity user to obtain personal distributions.

Figure SY-1 shows the relationships between user passwords, AUT, and the user profiles.



Note: P=password, N=name so that it can be found by name or by password.

Figure SY-1. Authorized User Table/User Profile Relationship

Object Authorization

Because all of the functions and the data available on the system exist as objects, their use can be controlled by authorizing system users to use them. Object authorization is the process of controlling which individuals are authorized to use an object and of assigning the rights of use each individual has in relation to that object. Authorization to use an object is usually administered by the owner of the object (the creator) and is enforced by the security component when any attempt is made to use the object.

The requirement that each object known to the system must be named and that each object must have an identified owner is basic to the concept of object authorization. The initial owner of an object is defined as the individual who created that object. Ownership of an object lets that individual authorize other users the right to use the object, transfer ownership of the object to another user, and display the authorized users and their rights of use for his object.

User authorization is the method by which the owner of an object can specify the users that can use his object. The owner of an object can authorize the use of his object in the following ways:

- Private: Only the owner of an object can use that object.
- Explicit: The owner of the object can identify, by user name, the system users that can use the object.
- Public: The owner of the object can specify that all system users are authorized to use the object. Public authority is kept with the object rather than being put in each user profile.

Rights of Use

How a user can use an object depends on what rights of use are included in the authority of the user. The owner of an object and the system security officer have all rights to the use of an object. Other system users can be granted some or all of the rights of use either through public authority or explicitly granted authority.

The rights of use supported by security and a brief description of those rights are as follows:

Object Rights: These rights apply to an object in its entirety. These rights are applicable to all objects and control the major functions available in the system at an object level.

- Object existence: This right provides the authority to control object ownership and existence.
- Object management: This right provides the authority to manage access and availability of objects.
- Operational: This right provides the authority to look at the description of an object and operate with the object. If additional data rights apply to an object, they further control what operations can occur. For example, to compile a program using an externally described data base file, only operational authority is required to the file. To actually read data from the file, read authority is also required.

Data Rights: These rights apply to the contents of an object. These rights are only applicable to objects that contain elements of information.

- Read: This right provides the authority to retrieve or materialize the contents of an object entry.
- Update: This right provides the authority to modify or replace an object entry.
- Add: This right provides the authority to add an object entry.
- Delete: This right provides the authorization to destroy or remove an object entry.

Security Modules

The security component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QSYUP–Create/Change User Profile (CRTUSRPRF, CHGUSRPRF)[1]: This module is used to create or change a user profile.

-->QSYDSUP–Display User Profile (DSPUSRPRF)[1]: This module is used to display user profile information to the user.

-->QSYDLUP–Delete User Profile (DLTUSRPRF)[1]: This module is used to delete a user profile from the system.

-->QSYGRAUT–Grant Object Authority (GRTOBJAUT)[1]: This module is used to grant authority to use an object to a specific user, to the public, or from a reference object.

-->QSYDSAUT–Display Object Authority (DSPOBJAUT)[1]: This module is used to display who has what type of authority to the use of a specific object.

-->QSYRVAUT–Revoke Object Authority (RVKOBJAUT)[1]: This module is used to take object authority away from a user or the public.

-->QSYCHONR–Change Object Owner (CHGOBJOWN)[1]: This module is used to change the ownership of an object from one user to another user.

QSYPGMCH–Program Check For Adopted Profile: This module checks program objects to see if they adopt their owner's user profile. This module is called by QSYCHONR.

QSYRVKDR–Revoke Data Rights: This module revokes data rights from the new owner of a data base logical file, since data rights have no meaning for a logical file. This module is called by QSYCHONR.

-->QSYDSUSR–Display Authorized Users (DSPAUTUSR)[1]: This module is used to display the contents of the AUT (passwords and user names).

-->QSYGRUSR–Grant User Authority (GRTUSRAUT)[1]: This module is used to grant the authority of one user profile to another user profile.

-->QSYVERFY–System Entry Verification: This module verifies the authority of the user to the system and its resources.

-->QSYAULIB–Retrieve Authorized Libraries: This module gets a list of all the libraries to which the process has *READ authority.

-->QSYCNV–Convert Authority: This module converts the CPF authorization keywords to bit strings.

-->QSYSVAUT–Save Authorized User Table: This module extracts the user name and password entries from the authorized user table and saves them.

-->QSYRSAUT–Restore Authorized User Table: This module restores the authorized user table by inserting the user name and password entries into it.

-->QSYHNAUT–Handle Authority Violation: This module provides a standard logging of CPF-detected authorization violations to the system history file and optionally signals an authorization violation exception.

-->QSYGRTSA–Grant Same Authority: This module is used to authorize an internal object the same as another internal object on the system.

-->QSYAUTEV–Authorization Event Handler: When an authorization violation occurs, this module handles the various authorization events that can be signaled and logs their occurrence to the system history file.

-->QSYGRSME–Grant Same Authority: This module is used to authorize an external object the same as another external object on the system.

[1]This module is a CPP (command processing program).

-->QSYACCIP–Access Interactive Profile: This module copies the function level entries specified from the permanent interactive profile in QSYS library to the temporary interactive profile in QTEMP library.

QSYCPYIP–Copy Interactive Profile: This module copies the permanent entries in the temporary interactive profile in QTEMP library to the permanent interactive profile in QSYS library and deletes the temporary interactive profile.

QSYCRTIP–Create Interactive Profile: This module creates the permanent interactive profile in QSYS and the temporary interactive profile in QTEMP.

-->QSYCHGIP–Change Interactive Profile Entry: This module changes the specified entry in the temporary interactive profile in QTEMP library.

-->QSYEXGRM–Extract Group Members: This module extracts the names and numbers of group members for the associated user profile.

-->QSYRMVIP–Remove Interactive Profile Entry: This module tags an entry in the temporary interactive profile in QTEMP to indicate the corresponding entry in the permanent interactive profile in QSYS is to be deleted.

-->QSYRTVIP–Retrieve Interactive Profile Entry: This module retrieves an entry from the temporary interactive profile in QTEMP.

-->QSYCKCMD–Check Command Authority: This module checks to see that the process is authorized to the command whose function is being requested from a display or menu.

-->QSYEXUNP–Extract User Name and Password: This module retrieves the process user profile name from the work control block table entry and its associated password from the authorized user table.

-->QSYRVSPC–Revoke Space Authority: This module revokes space authority from the old owner of a composite object when that object's ownership is changed.

| QSYUPASR–User Profile Associated Space Recovery: This module does recovery from a damaged or missing user profile associated space and assigns IBM-supplied defaults to the user profile.

-->QSYRTVUP–Retrieve User Profile Values: This module retrieves the values of the specified user profile attributes and places them into CL variables in a CL program.

Create User Profile and Change User Profile Commands

Figure SY-2 and the following text describe the operation of a Create User Profile (CRTUSRPRF) command and a Change User Profile (CHGUSRPRF) command.

1 The command analyzer decodes either a Create User Profile command or a Change User Profile command, and control is transferred to QSYUP.

2 The AUT (authorized user table) is checked for a duplicate password. If there is not a duplicate password, the appropriate entries are added or changed.

3 The user profile is created or changed.

4 An entry in the OIR (object information repository) is added or changed for the QSYS library.

5 Control is returned to the caller.

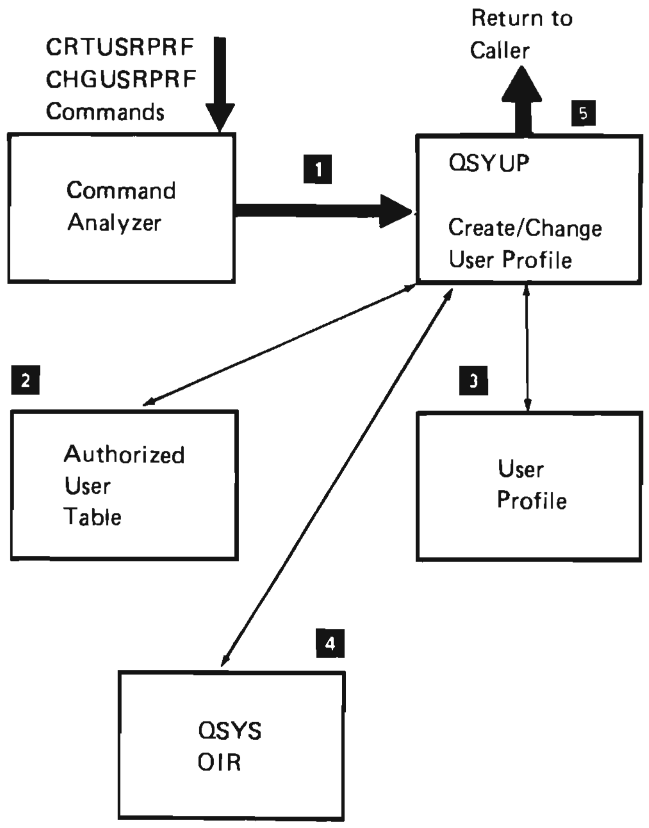


Figure SY-2. Create/Change User Profile Command Overview

Display User Profile Command

Figure SY-3 and the following text describe the operation of the Display User Profile (DSPUSRPRF) command.

1 The command analyzer decodes a Display User Profile command and control is transferred to QSYDSUP.

2 The required information is retrieved from the AUT, the user profile, and the OIR for the QSYS library. The information can be printed or displayed.

3 When the information is displayed, the user (security officer only) may press the CF3 key from the *BASIC display to change the user profile. QSYUP is called via the command analyzer to change the user profile.

4 Control is returned to the caller.

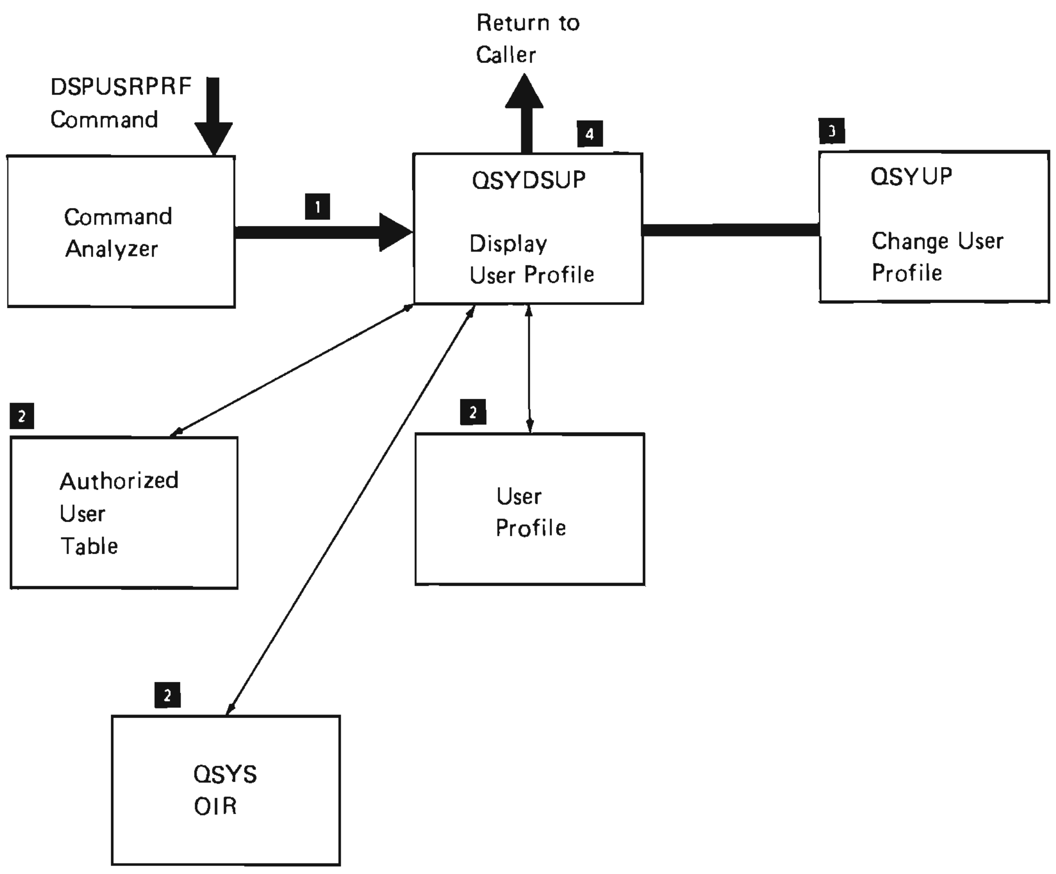


Figure SY-3. Display User Profile Command Overview

User Profile Associated Space Recovery

Figure SY-4 and the following text describe the operation of the user profile associated space recovery function.

1 QSYUPASR is called via the ?EXTUPAS or ?CHGUPAS macro.

2 A new user profile associated space is appended to the user profile and the associated space is initialized with IBM-supplied defaults.

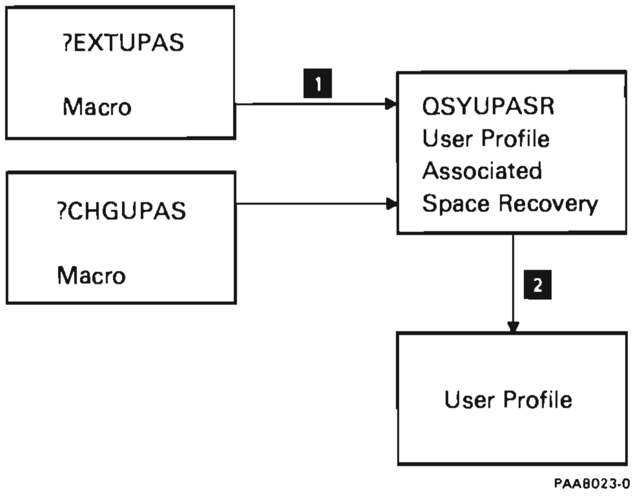


Figure SY-4. User Profile Associated Space Recovery

Retrieve User Profile Command

Figure SY-5 and the following text describe the operation of a Retrieve User Profile (RTVUSRPRF) command. The RTVUSRPRF command can only be executed from a CL program.

1 The command analyzer decodes a Retrieve User Profile command and control is transferred to QSYRTVUP, or the module is called by the ?EXTUPI macro.

2 The required information is retrieved from the AUT, the user profile, and the OIR for the QSYS library.

3 The requested information is then returned to CL variables in a CL program.

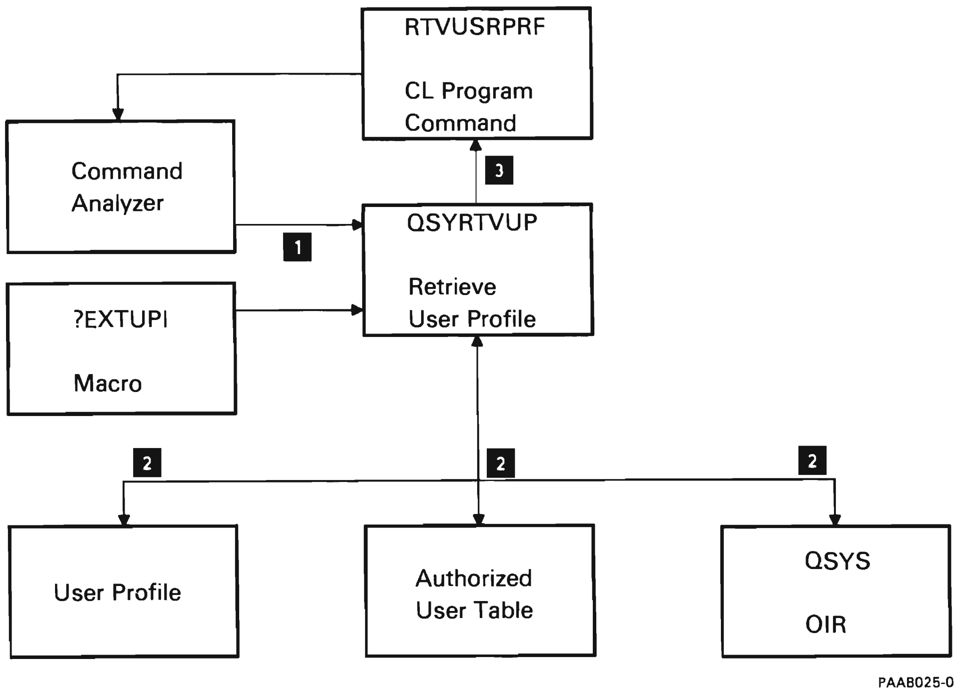


Figure SY-5. Retrieve User Profile Command Overview

Delete User Profile Command

Figure SY-6 and the following text describe the operation of a Delete User Profile (DLTUSRPRF) command.

1 The command analyzer decodes a Delete User Profile command and control is transferred to QSYDLUP.

2 The user profile is deleted.

3 The password and user name entries are deleted from the AUT (authorized user table).

4 Delete the interactive profile associated with this user profile, if one exists.

5 Information relating to the user profile is deleted from the OIR of the QSYS library.

6 Control is returned to the caller.

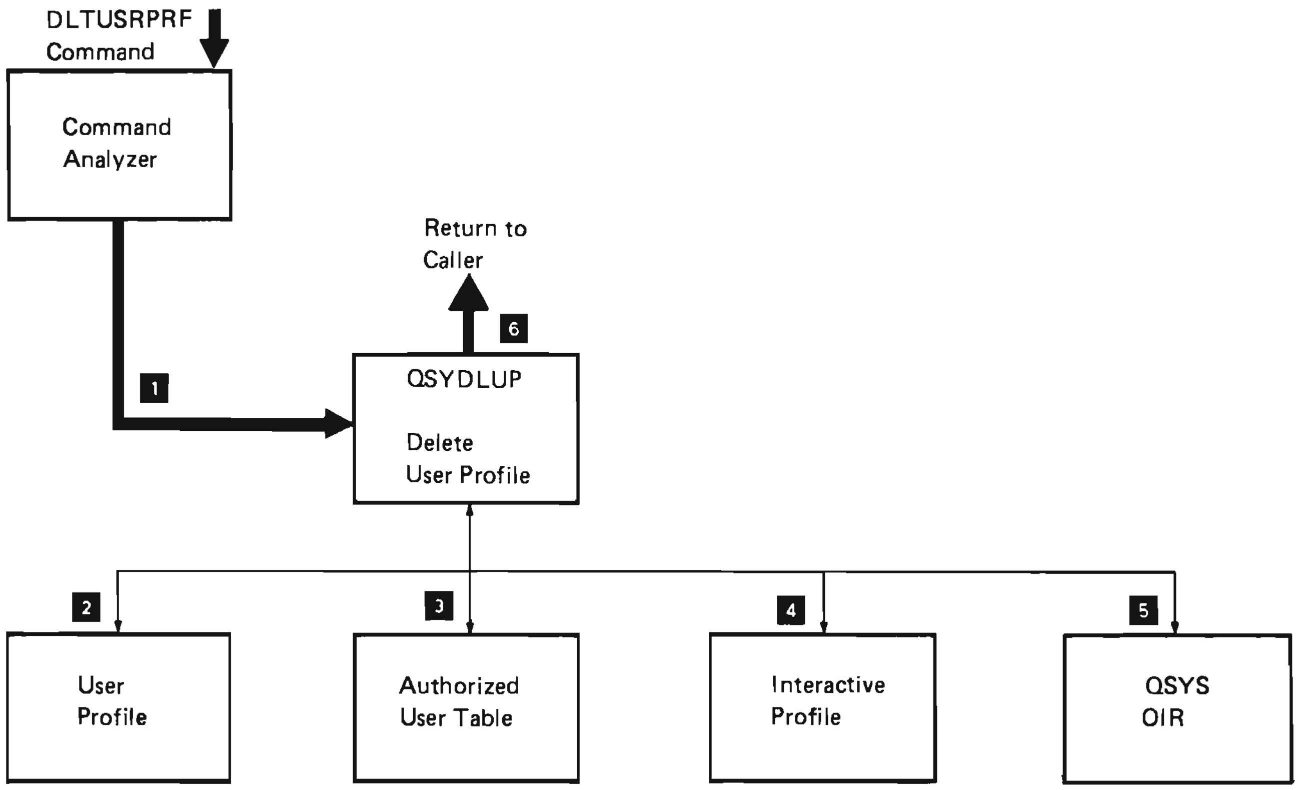


Figure SY-6. Delete User Profile Command Overview

Grant Object Authority Command

Figure SY-7 and the following text describe the operation of a Grant Object Authority (GRTOBJAUT) command.

1 The command analyzer decodes a Grant Object Authority command and control is transferred to QSYGRAUT, or the module is called by the ?GRTOBJ macro.

2 The grant GFDT (generic function definition table) is checked to determine function applicability, object structure, and the routine to be called to perform the function.

3 QLILIST is called to locate the names and addresses of the objects to be authorized.

4 If authority is being granted from a reference object, the authorized users of the reference object are materialized.

5 The QDMROUTE module of common data management is called if the object type for the object to which authority is being granted is FILE.

6 If public authority is being granted, it is kept with the object.

7 If explicit authority is being granted, an authorized object entry is made in each user profile of the users being authorized. An authorized user entry is also made in the user profile of the user owning the object for each user authorized to use the object.

8 Control is returned to the caller.

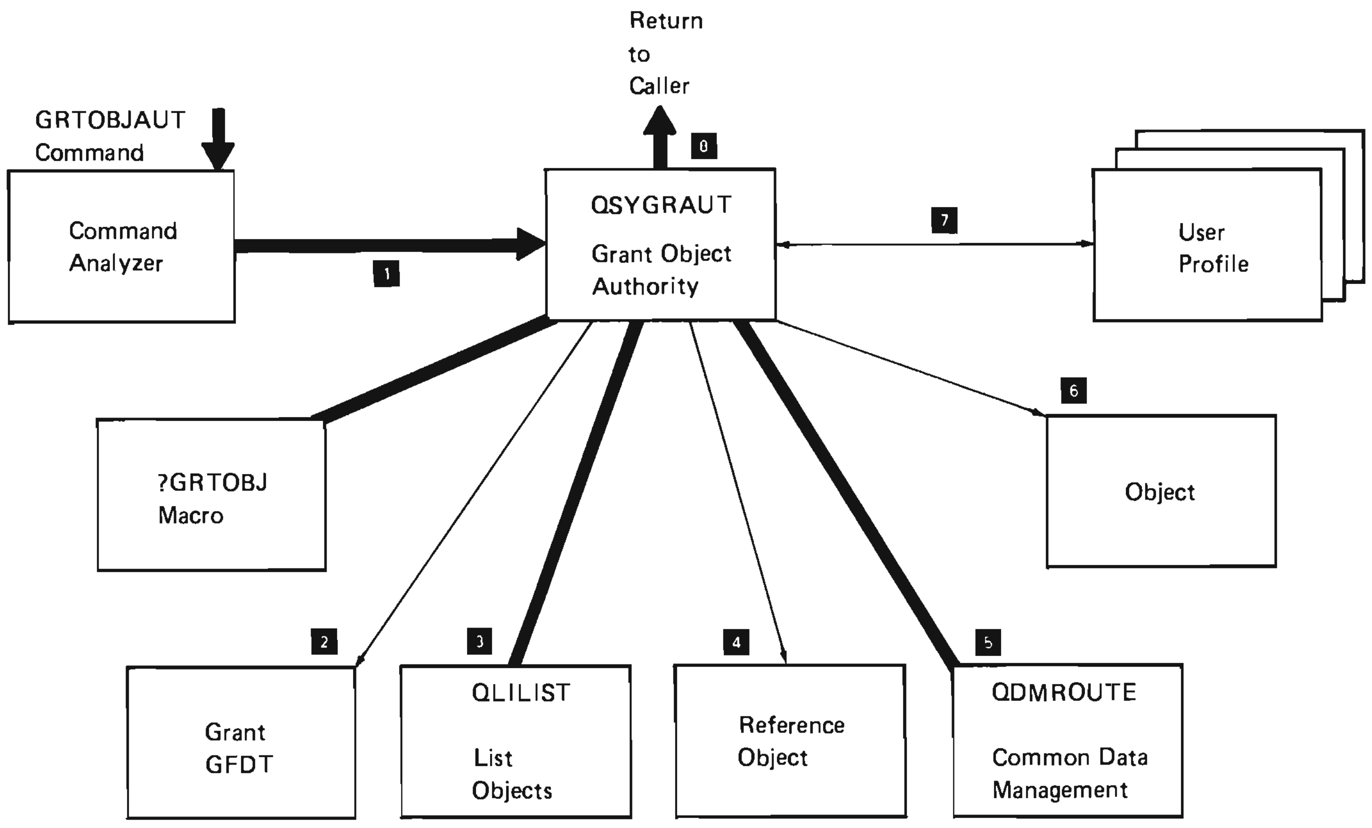


Figure SY-7. Grant Object Authority Command Overview

Display Object Authority Command

Figure SY-8 and the following text describe the operation of a Display Object Authority (DSPOBJAUT) command.

1 The command analyzer decodes a Display Object Authority command and control is transferred to QSYDSAUT.

2 The public authority for the object is retrieved from the object and the explicit authority of the authorized users is retrieved from the user profile owning the object. The information can be either displayed or printed.

3 The user with *OBJMGT authority for the object may press the CF3 key to change the authority for the displayed object. QSYGRAUT is called to grant authority. QSYRVAUT is called to revoke authority.

4 Control is returned to the caller.

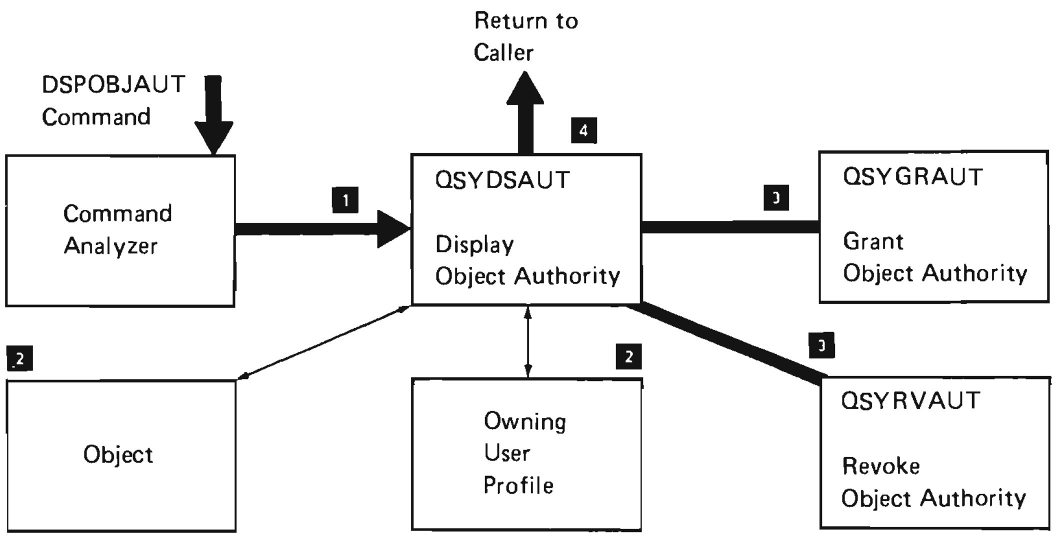


Figure SY-8. Display Object Authority Command Overview

Revoke Object Authority Command

Figure SY-9 and the following text describe the operation of a Revoke Object Authority (RVKOBJAUT) command.

1 The command analyzer decodes a Revoke Object Authority command and control is transferred to QSYRVAUT, or the module is called by the ?RVKOBJ macro.

2 The revoke GFDT (generic function definition table) is checked to determine the function applicability, object structure, and the routine to be called to perform the function.

3 QLILIST is called to locate the names and addresses of the objects to have authority revoked.

4 The QDMROUTE module of common data management is called if the object type for which authority is to be revoked is FILE.

5 The QSPHNSPQ module of spooling is called if the object type for which authority is to be revoked is JOBQ or OUTQ.

6 If public authority is being revoked, it is removed from the object.

7 If explicit authority is being revoked, the authorized object entry for the object is removed from the profile of each user being revoked and the authorized user entry for each user being revoked is removed from the user profile owning the object.

8 Control is returned to the caller.

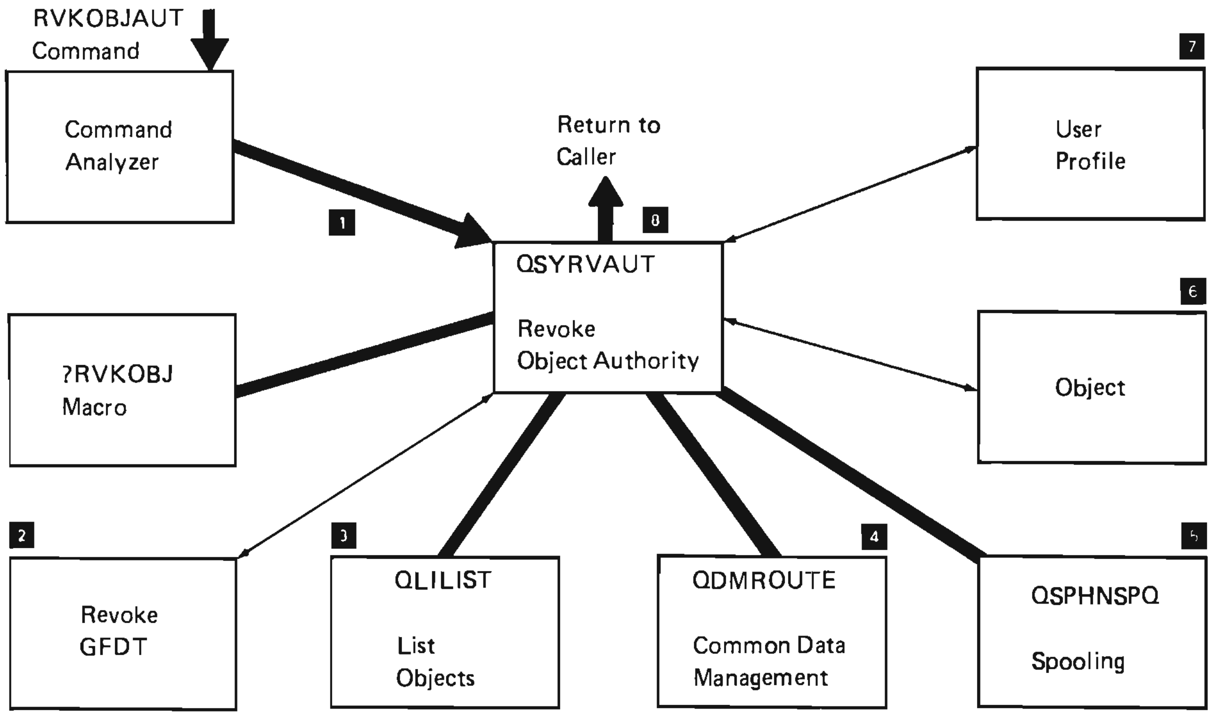


Figure SY-9. Revoke Object Authority Command Overview

Change Object Owner Command

Figure SY-10 and the following text describe the operation of a Change Object Owner (CHGOBJOWN) command.

1 The command analyzer decodes a Change Object Owner command and control is transferred to QSYCHONR, or the module is called by the ?CHGONR macro.

2 The transfer GFDT (generic function definition table) is checked to determine function applicability, object structure, and the routine to be called to perform the function.

3 The QDMROUTE module of common data management is called if the object type of the object for which ownership is being changed is FILE.

4 QSYPGMCH is called to determine if the program adopts its owner's user profile, if object type PGM is the object type for which ownership is being changed. Only the security officer can change ownership of a program that adopts its owner's profile.

5 QSYRVSPC is called to revoke space authority from the old owner of the object for which ownership is being changed.

6 The name of the object owner is changed in the object.

7 The owned object entry is removed from the current owner and is put in the user profile of the new owner.

8 Control is returned to the caller.

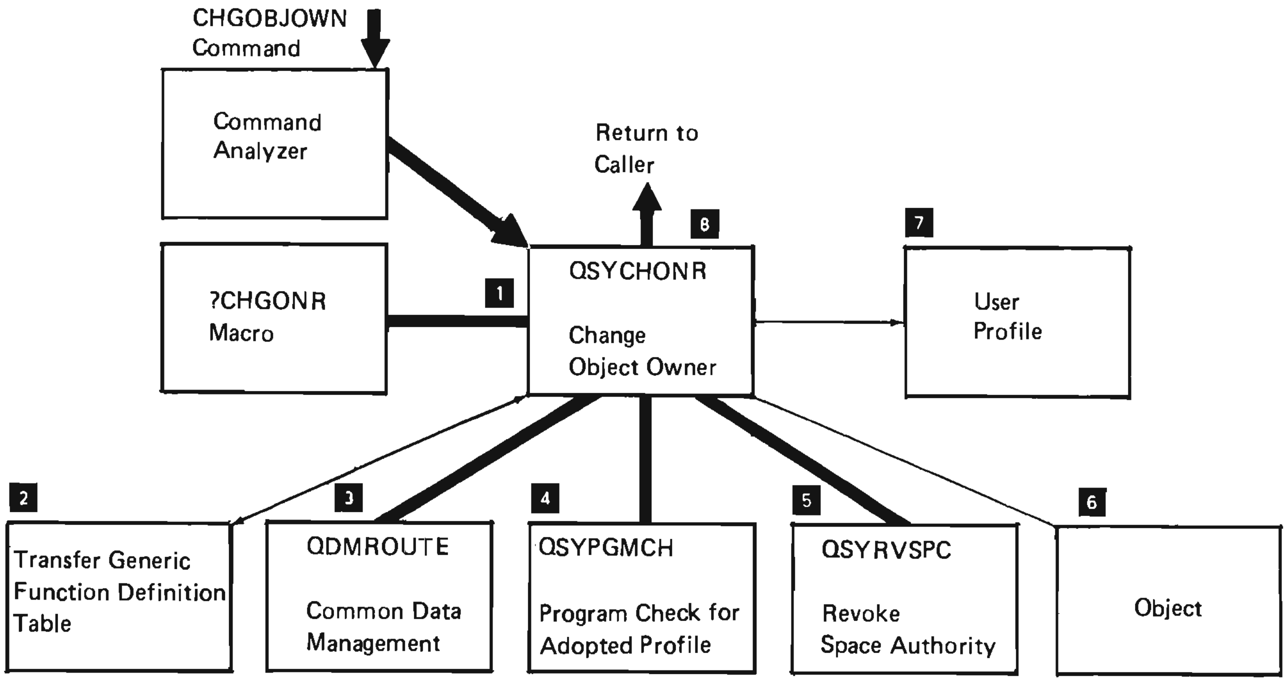


Figure SY-10. Change Object Owner Command Overview

Grant User Authority Command

Figure SY-11 and the following text describe the operation of a Grant User Authority (GRTUSRAUT) command.

1 The command analyzer decodes a Grant User Authority command and control is transferred to QSYGRUSR.

2 An authorized object entry is made in each user profile of the users being authorized. An authorized user entry is also made in the user profile of the user owning the object for each user authorized to use the object.

3 Control is returned to the caller.

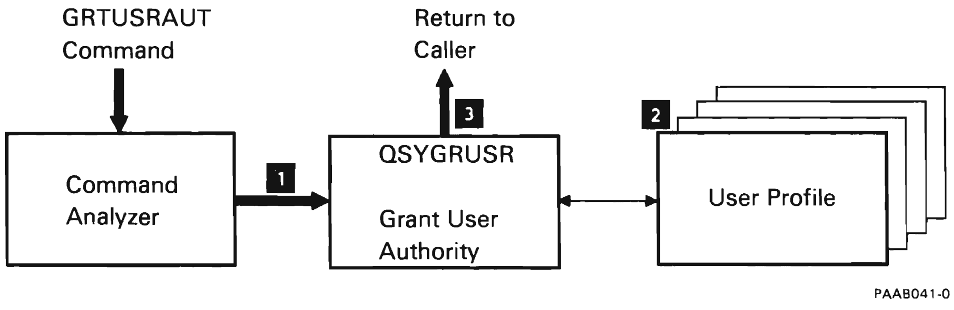


Figure SY-11. Grant User Authority Command Overview

Display Authorized Users Command

Figure SY-12 and the following text describe the operation of a Display Authorized Users (DSPAUTUSR) command.

1 The command analyzer decodes a Display Authorized Users command and control is transferred to QSYDSUSR.

2 The entries are retrieved from the AUT (authorized user table). The user names and their associated passwords, along with an indication whether the user profile is a group profile, are displayed or printed in alphabetical order, either by user name or password.

3 Control is returned to the caller.

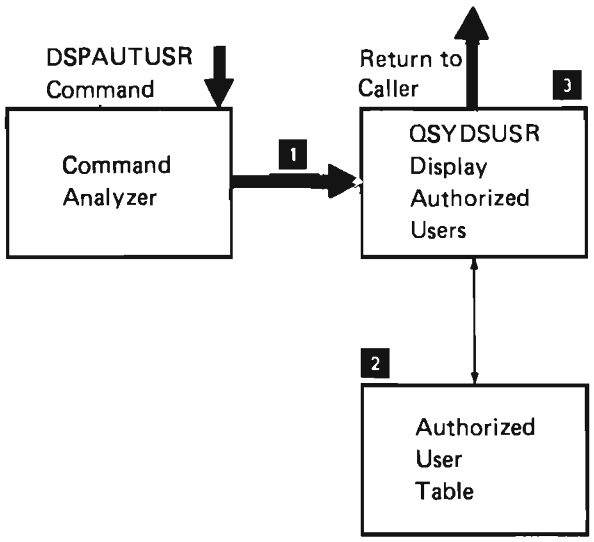


Figure SY-12. Display Authorized Users Command Overview

Verify System Entry Authorization

Figure SY-13 and the following text describe the operation of the verify system entry authority function.

1 QSYVERFY is called by the ?VERIFY macro.

2 The AUT (authorized user table) is checked to:

- Verify the validity of the password
- Retrieve the user profile name, the pointer to the user profile, and the priority limit of the user

3 The following objects are checked to see if the user is authorized to use them (explicitly or publicly):

- Job description
- Job input queue and the library in which it resides
- Output queue and the library in which it resides
- Subsystem description
- Work station
- Message queue and the library in which it resides
- Libraries on the library list

4 The following indicators are returned to the caller of the ?VERIFY macro:

- User password found
- User name found
- Authorized to use the job description
- Authorized to use the job input queue
- Authorized to use the subsystem description
- Authorized to use the work station
- Authorized to use the message queue
- Authorized to use the output queue
- Authorized to the libraries on the library list

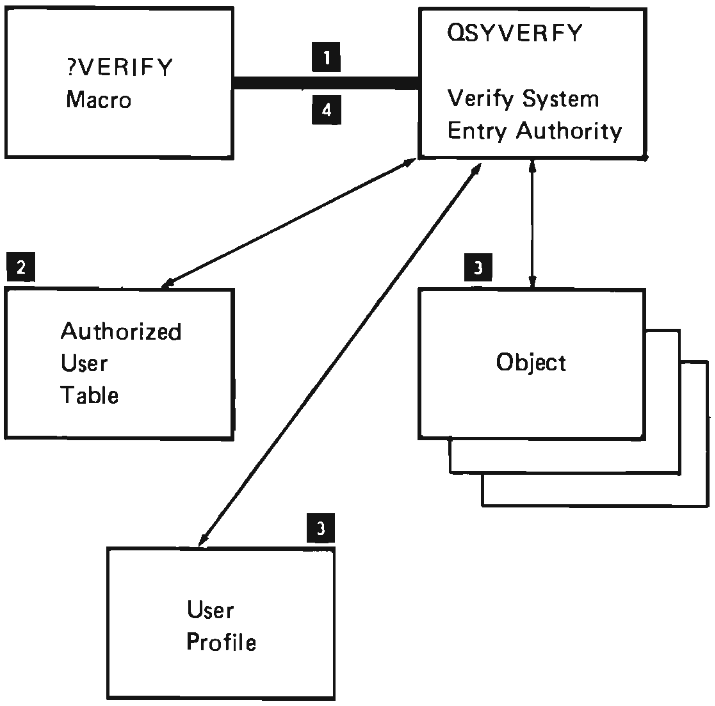


Figure SY-13. Verify System Entry Authority Overview

Retrieve Authorized Libraries

Figure SY-14 and the following text describe the retrieve authorized libraries function.

1 QSYAULIB is called by the ?RTVALIB macro.

2 All of the libraries are retrieved from the machine context.

3 Each library is checked to see if the authority to use it has been publicly granted.

4 If the authority to use a library has not been publicly authorized, the process user profile and any adopted user profiles are checked for explicit authority to use the library.

5 A list of pointers to *all* of the libraries that the process is authorized to use (either public or explicit *READ authority) is returned to the caller of the macro.

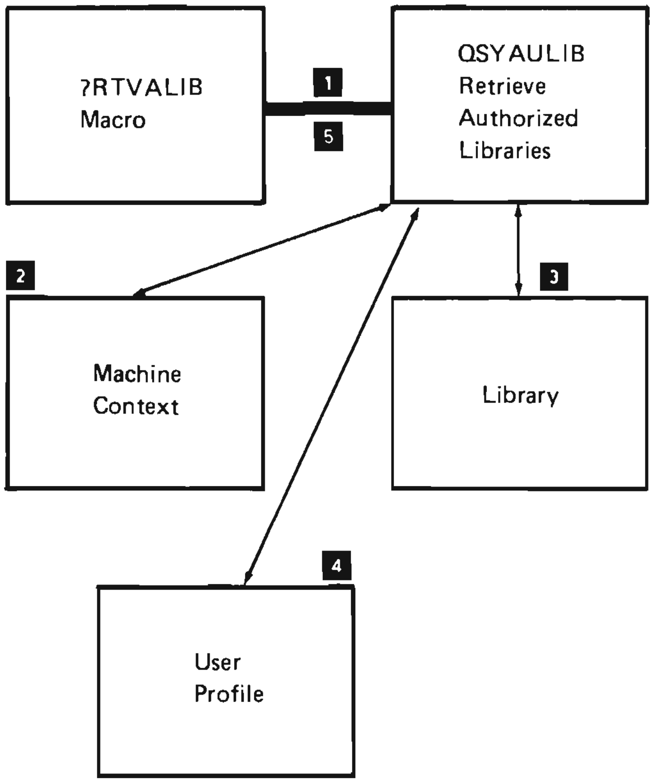


Figure SY-14. Retrieve Authorized Libraries Overview

Convert Authority

Figure SY-15 and the following text describe the operation of the convert authority function.

1 QSYCNV is called by the ?CNVAUT macro.

2 QSYCNV converts CPF authorization keywords to their equivalent machine interface bit strings.

3 The bit string representation of the authorization keyword(s) is returned to the caller of the macro.

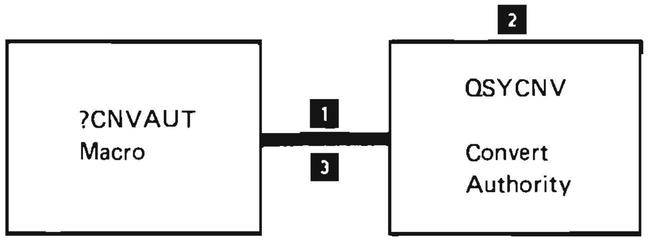


Figure SY-15. Convert Authority Function

Save Authorized User Table

Figure SY-16 and the following text describe the operation of the save authorized user table function.

1 QSYSVAUT is called by QSRSVOBJ as part of save system.

2 All of the user name and password entries are extracted from the AUT (authorized user table) and saved in a space.

3 A space containing all the AUT entries is returned to the caller of the module.

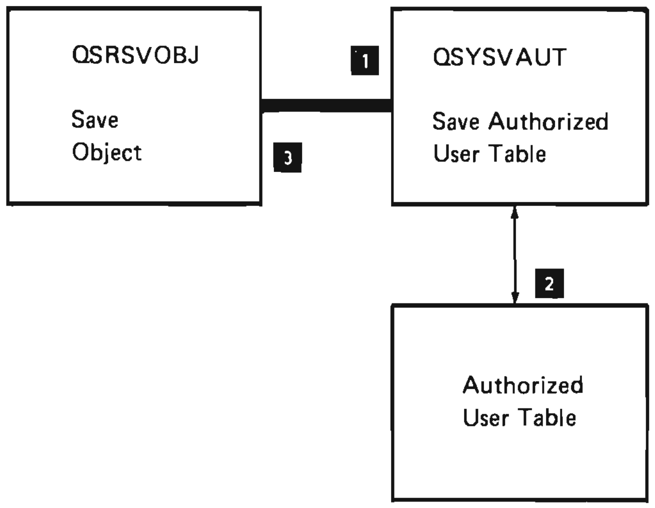


Figure SY-16. Save Authorized User Table Function

Restore Authorized User Table

Figure SY-17 and the following text describe the operation of the restore authorized user table function.

1 QSYRSAUT is called by QSRRSOBJ as part of restore system.

2 All of the user name and password entries are inserted into the AUT (authorized user table) unless they already exist. The user profiles are resolved to and the pointers in the entries are updated.

3 Control is returned to the caller.

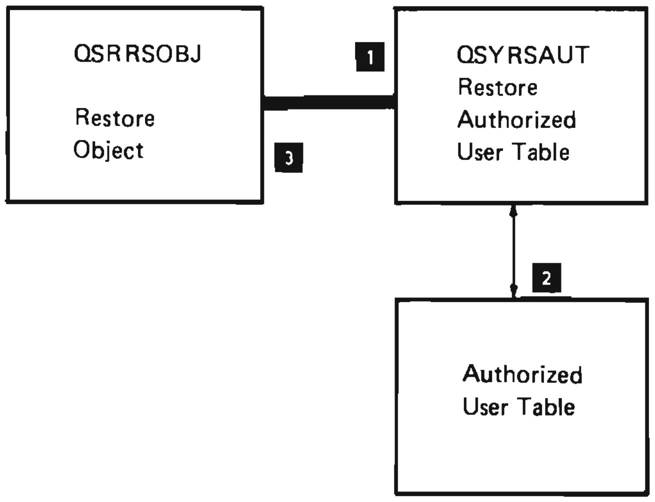


Figure SY-17. Restore Authorized User Table Function

Handle Authority Violation

Figure SY-18 and the following text describe the operation of the handle authority violation function.

1 QSYHNAUT is called by the ?CHKAUT, ?SECLOG, or ?EXTIPGM macro.

2 If QSYHNAUT is called by the ?CHKAUT or ?SECLOG macros, an authorization violation message is logged to the history file, and optionally the *user not authorized to object* exception (CPF2209) or the *user not authorized to nonobject* exception (CPF2206) is signaled.

3 If QSYHNAUT is called by the ?EXTIPGM macro, the initial program to be invoked following a successful logon is extracted from the AUT (authorized user table) for return to the caller.

4 Control is returned to the caller.

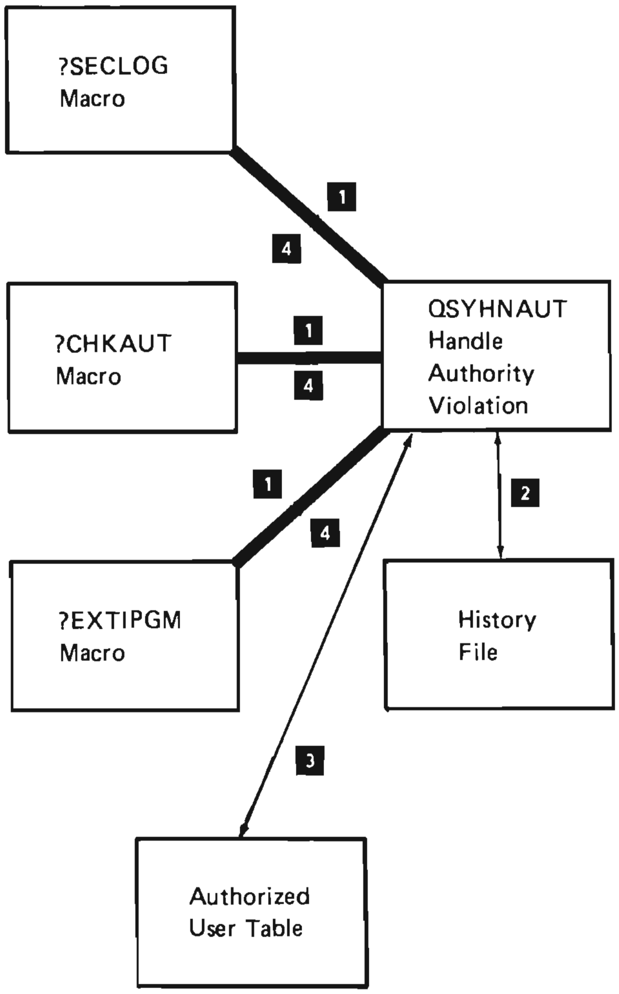


Figure SY-18. Handle Authority Violation Function

Authorization Event Handler

Figure SY-19 and the following text describe the operation of the authorization event handler function.

1 QSYAUTEV is invoked in the system arbiter process: whenever the machine interface detects an authorization violation, it signals an authorization event (hex 0002 0000).

2 The following authorization violation messages are logged to the history file depending on the event signaled:

Event	**Message**
0002 0101	JOB__________NOT AUTHORIZED TO OBJECT __________TYPE __________(CPF2218)
0002 0201	JOB__________NOT AUTHORIZED TO PRIVILEGED INSTRUCTION (CPF2219)
0002 0301	JOB__________DOES NOT HAVE SPECIAL AUTHORITY (CPF2220)

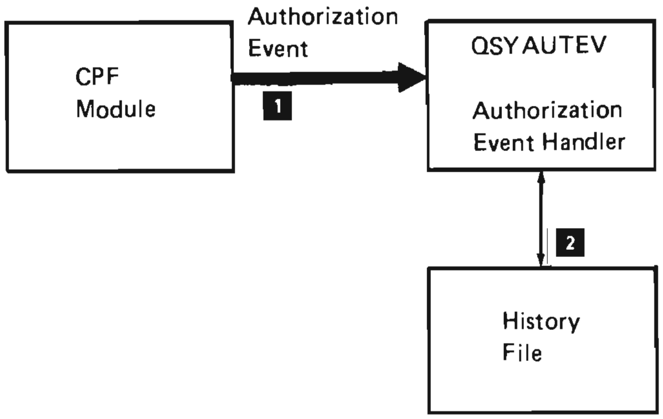


Figure SY-19. Authorization Event Handler Overview

Grant Same Authority

Figure SY-20 and the following text describe the operation of the grant same authority function.

1 QSYGRTSA is called by the ?GRTSAMEA macro.

2 Any existing authorizations for the object (new internal object) to be authorized are revoked.

3 The users of the old internal object and the authority they have for that object are materialized.

4 All of the authorized users of the old internal object are granted the same authority for the new internal object as they had for the old internal object.

5 Control is returned to the caller.

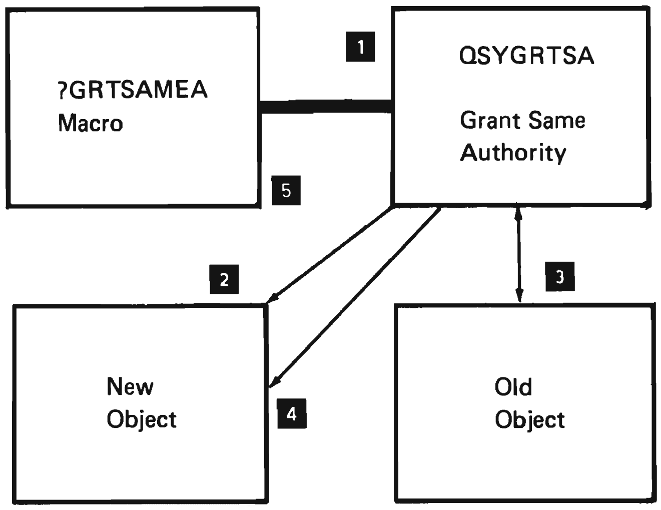


Figure SY-20. Grant Same Authority Overview

Grant Duplicate Authority

Figure SY-21 and the following text describe the operation of the grant duplicate authority function.

1 QSYGRSME is called by the ?GRTDUPAU macro.

2 Any authorization that currently exists for the new external object is revoked.

3 The authorized users of the old external object and the authorization they have for the object are materialized.

4 The authorized users of the old external object are granted the same authorization for the new external object as they had for the old external object.

5 Control is returned to the caller.

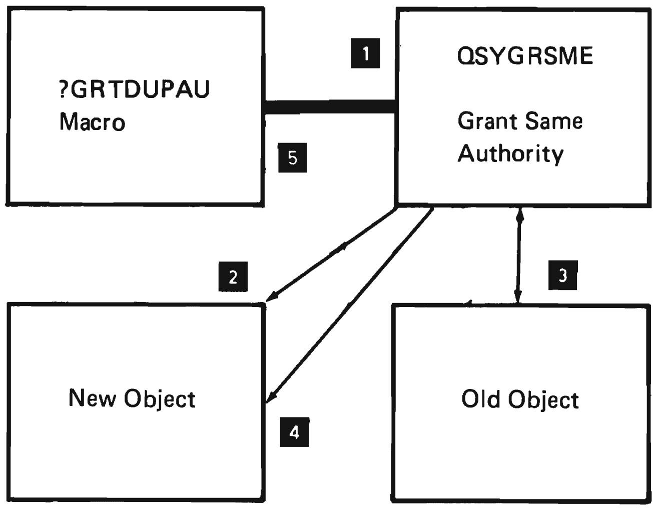


Figure SY-21. Grant Duplicate Authority Overview

Revoke Data Rights

Figure SY-22 and the following text describe the operation of the revoke data rights function.

1 QSYRVKDR is called by the ?TFROBJO macro if the transferred object is a data base logical file.

2 All data rights of use for the file are revoked from the user to whom the file ownership has just been transferred.

3 Control is returned to the caller.

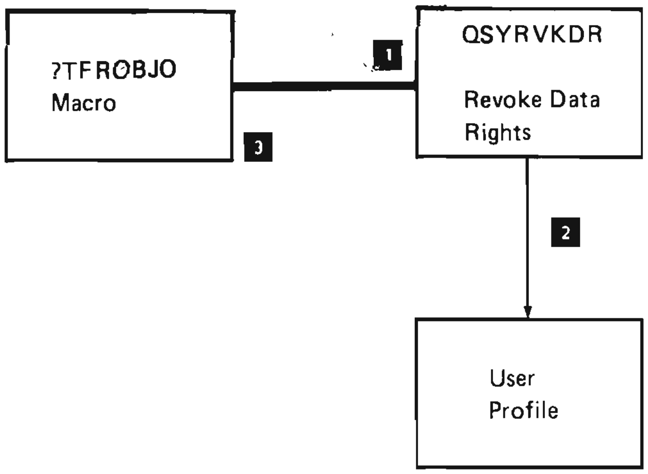


Figure SY-22. Revoke Data Rights Overview

Program Check for Adopted Profile

Figure SY-23 and the following text describe the operation of the program check for adopted profile function.

1 QSYPGMCH is called via the ?CALL macro.

2 The program is materialized to check if it adopts its owner's user profile.

3 An indicator as to whether the program adopts its owner's user profile or not is returned to the caller. As an alternative function, transfer ownership of the program to the group profile.

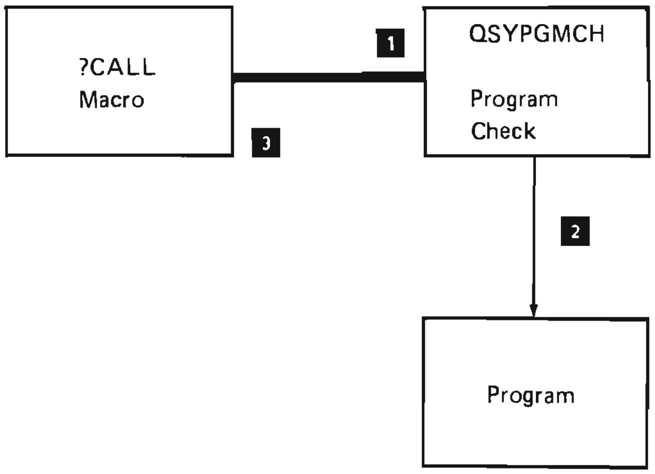


Figure SY-23. Program Check for Adopted Profile Overview

Access Interactive Profile

Figure SY-24 and the following text describe the operation of the access interactive profile function.

1 QSYACCIP is called via the ?ACCIP macro.

2 QSYCRTIP is called to create a permanent interactive profile in QSYS library and a temporary interactive profile in QTEMP library, if they do not exist.

3 The entries in the permanent interactive profile are copied to the temporary interactive profile.

4 A scope message, CPF2250, is sent to the caller of the module to have QSYCPYIP invoked when the caller's invocation is terminated. QSYCPYIP copies the temporary interactive profile (with additions, deletions, and changes) back to the permanent interactive profile.

5 Control is returned to the caller.

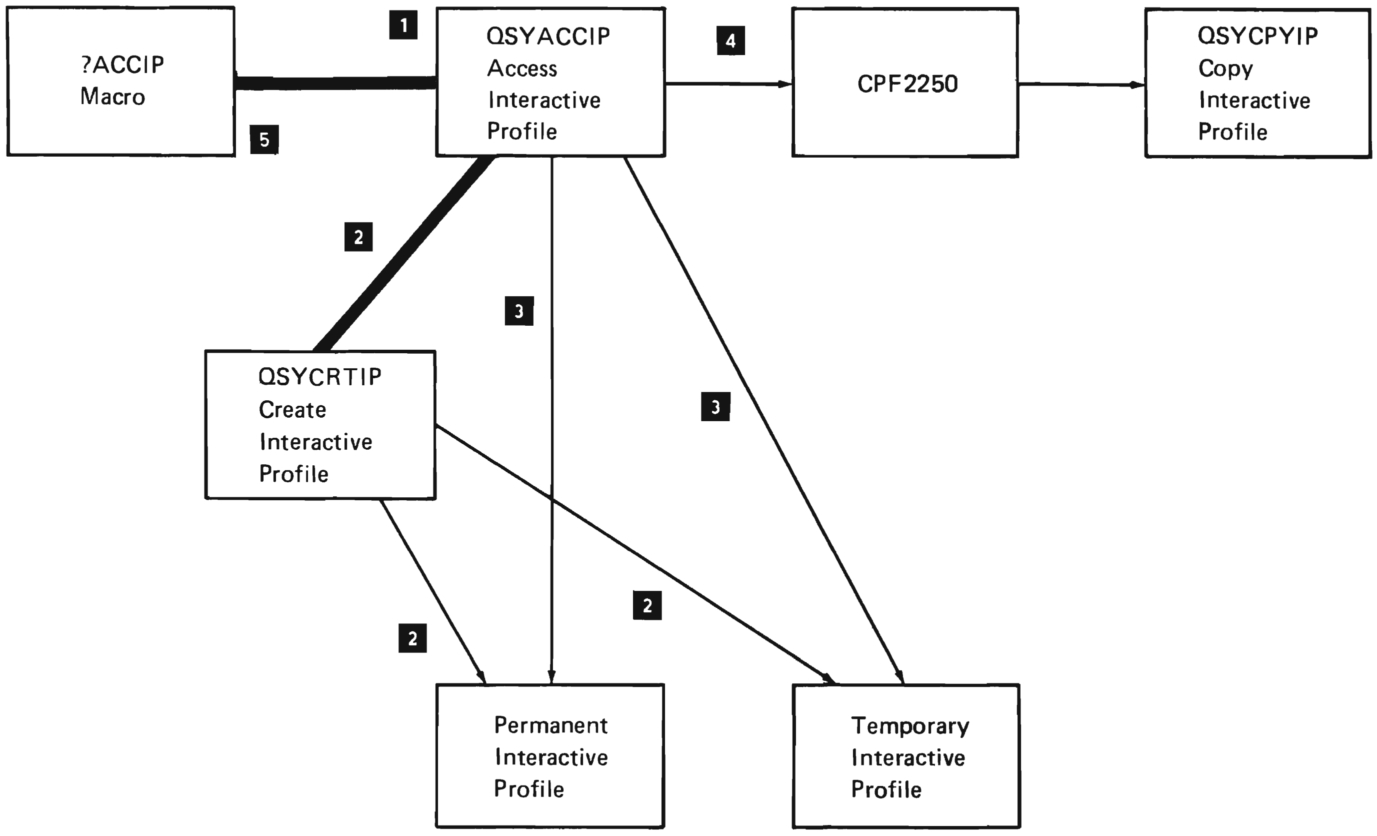

?ACCIP Macro
1
5
QSYACCIP Access Interactive Profile
4
CPF2250
QSYCPYIP Copy Interactive Profile
2
3
3
QSYCRTIP Create Interactive Profile
2
2
Permanent Interactive Profile
Temporary Interactive Profile

Figure SY-24. Access Interactive Profile Overview

Change Interactive Profile Entry

Figure SY-25 and the following text describe the operation of the change interactive profile entry function.

1 QSYCHGIP is called via the ?CHGIPE macro.

2 If the temporary interactive profile does not exist in QTEMP, QSYACCIP is invoked via the ?ACCIP macro to create it.

3 QSYCHGIP changes the specified entry in the QTEMP library.

4 Control is returned to the caller.

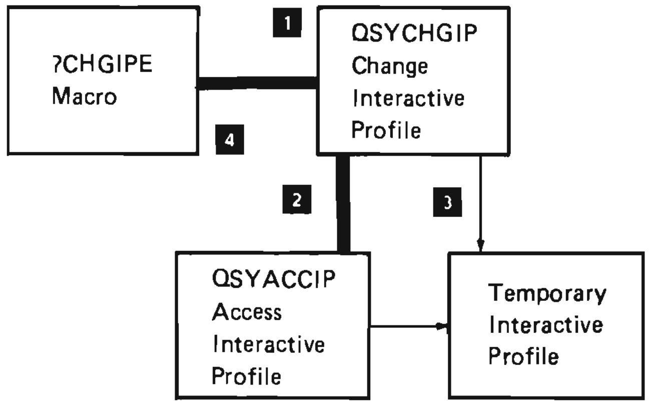

?CHGIPE Macro
1
4
QSYCHGIP Change Interactive Profile
2
3
QSYACCIP Access Interactive Profile
Temporary Interactive Profile

Figure SY-25. Change Interactive Profile Entry Overview

Copy Interactive Profile

Figure SY-26 and the following text describe the operation of the copy interactive profile function.

1 QSYCPYIP is invoked when the invocation that called QSYACCIP is terminated. QSYCPYIP is called as a result of the scope message CPF2250 sent from QSYACCIP to its caller.

2 If the permanent interactive profile in the QSYS library cannot be found or is damaged, QSYCRTIP is called to create it.

3 The permanent entries in the temporary interactive profile are copied into the permanent interactive profile, and the temporary interactive profile is deleted from QTEMP.

4 Control is returned to the caller.

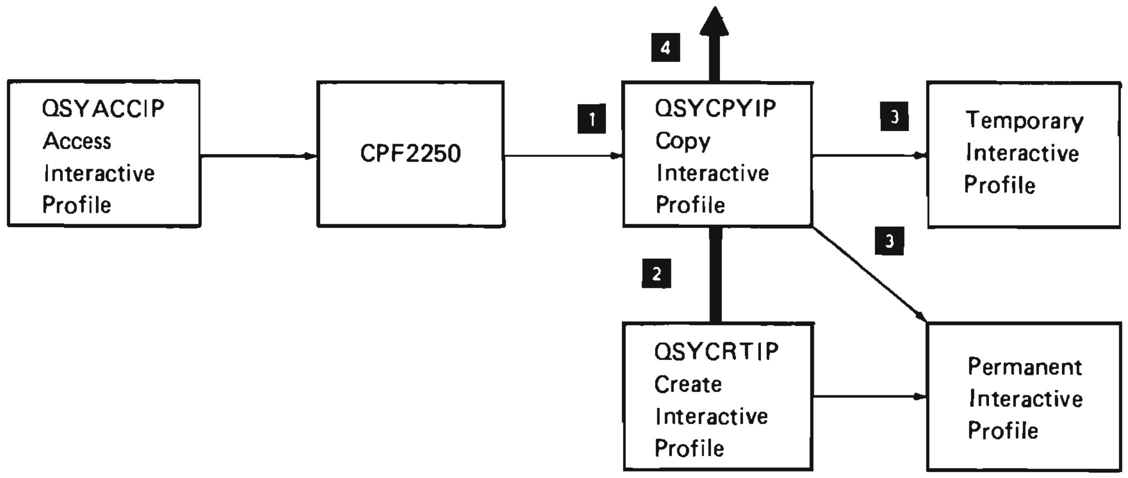

QSYACCIP Access Interactive Profile
CPF2250
1
4
QSYCPYIP Copy Interactive Profile
3
Temporary Interactive Profile
3
2
QSYCRTIP Create Interactive Profile
Permanent Interactive Profile

Figure SY-26. Copy Interactive Profile Overview

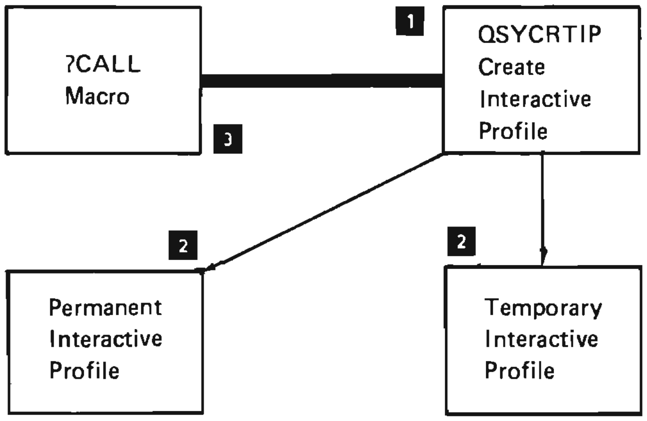
Create Interactive Profile

Figure SY-27 and the following text describe the operation of the create interactive profile function.

1 QSYCRTIP is called via the ?CALL macro.

2 QSYCRTIP creates the permanent interactive profile in the QSYS library if it does not exist and transfers ownership of it to the security officer, and/or creates the temporary interactive profile in the QTEMP library if it does not exist.

3 Control is returned to the caller.



?CALL Macro
1
3
QSYCRTIP Create Interactive Profile
2
2
Permanent Interactive Profile
Temporary Interactive Profile

Figure SY-27. Create Interactive Profile Overview

Remove Interactive Profile Entry

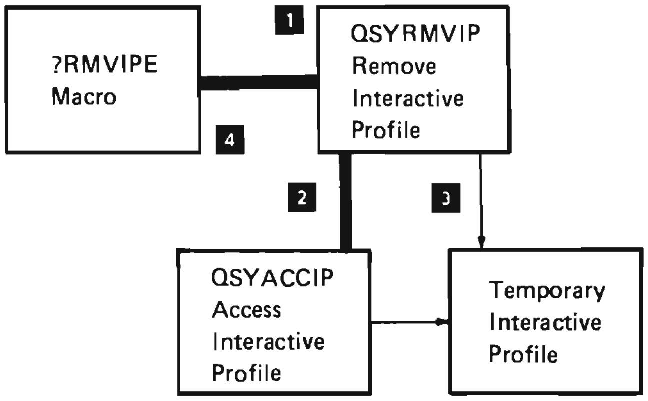
Figure SY-28 and the following text describe the operation of the remove interactive profile entry function.

1 QSYRMVIP is called via the ?RMVIPE macro.

2 If the temporary interactive profile in QTEMP library does not exist, QSYACCIP is invoked via the ?ACCIP macro to create it.

3 QSYRMVIP tags an entry in the temporary interactive profile for deletion. The tag indicates the corresponding entry in the permanent interactive profile is to be deleted and therefore is not available for use.

4 Control is returned to the caller.



?RMVIPE Macro
1
4
QSYRMVIP Remove Interactive Profile
2
3
QSYACCIP Access Interactive Profile
Temporary Interactive Profile

Figure SY-28. Remove Interactive Profile Entry Overview

Retrieve Interactive Profile Entry

Figure SY-29 and the following text describe the operation of the retrieve interactive profile entry function.

1 QSYRTVIP is called via the ?RTVIPE macro.

2 If the temporary interactive profile in QTEMP library does not exist, QSYACCIP is invoked via the ?ACCIP macro to create it.

3 The specified entry is retrieved from the temporary interactive profile.

4 Control is returned to the caller.

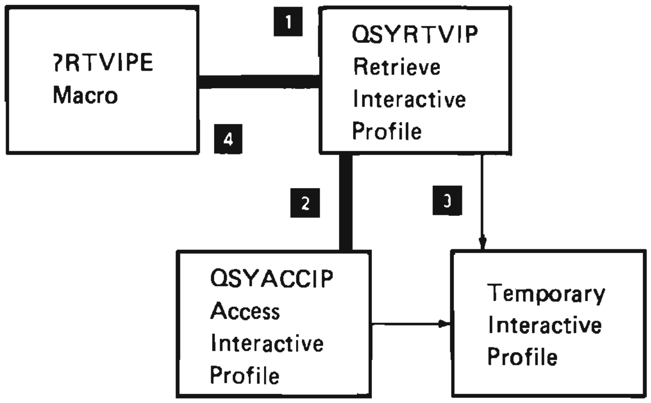


Figure SY-29. Retrieve Interactive Profile Entry Overview

Check Command Authority

Figure SY-30 and the following text describe the operation of the check command authority function.

1 QSYCKCMD is called via the ?CHKCMDA macro.

2 A check is made to see if the process is authorized to the command. The public authority, process user profile, and adopted user profile are also checked for proper authority.

3 An indicator is returned to the caller of the macro that states whether the process is authorized to the command or not.

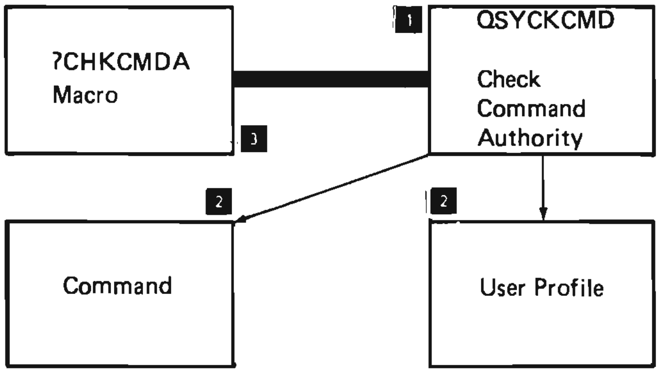


Figure SY-30. Check Command Authority Overview

Extract User Name and Password

Figure SY-31 and the following text describe the operation of the extract user name and password function.

1. QSYEXUNP is called via the ?EXTUNAP macro.
2. The process user profile name is retrieved from the work control block.
3. The user profile name's associated password is retrieved from the authorized user table.
4. The user profile name and password are returned to the caller.

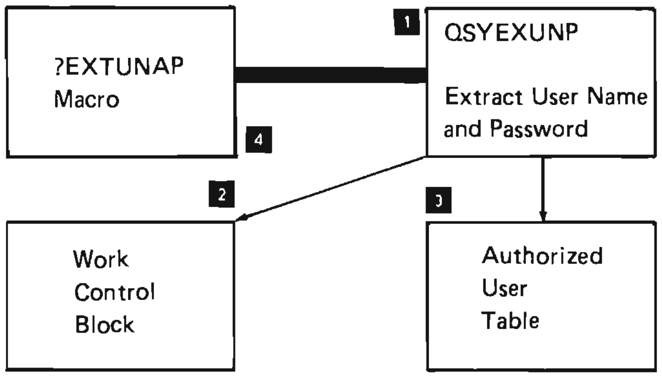


Figure SY-31. Extract User Name and Password Overview

Revoke Space Authority

Figure SY-32 and the following text describe the operation of the revoke space authority function.

1. QSYRVSPC is called via the ?CALL macro when a composite object changes ownership.
2. Space authority is revoked from the old owner of the object.
3. Control is returned to the caller.

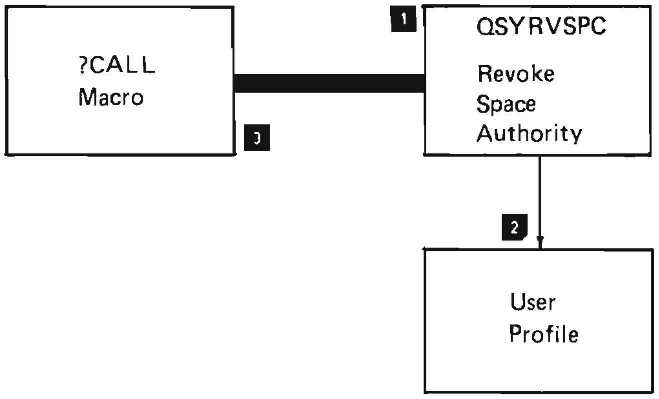


Figure SY-32. Revoke Space Authority Overview

Extract Group Members

Figure SY-33 and the following text describe the operation of the extract group members function.

1 QSYEXGRM is called via the ?EXTGRPM macro.

2 The number and names of the members of the group are retrieved from the authorized user table.

3 The number and names of the members of the group are returned to the caller.

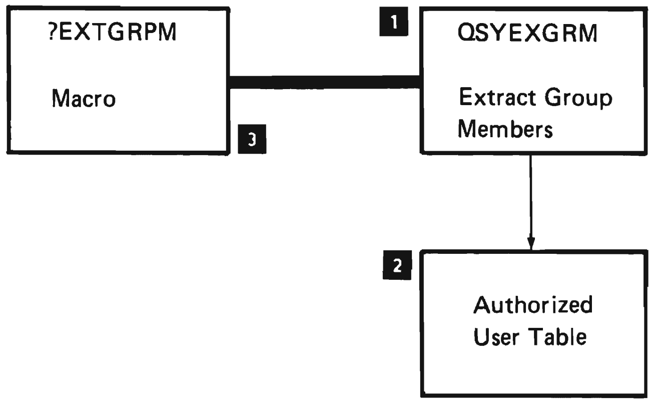


Figure SY-33. Extract Group Members

Tape Function Manager

INTRODUCTION

The tape function manager component of the CPF (control program facility) provides the support for the tape device on System/38.

The tape is a magnetic storage device that is supported as a system I/O device, data interchange device, and save/restore device. It contains a control unit and up to four drives.

The following tape functions are supported by the tape function manager:

- Initialize volume
- Display volume
- Dump volume
- Check volume
- Open tape file for processing
- Close tape file for processing
- Read data from a tape file
- Write data to a tape file
- End-of-volume processing

GENERAL OVERVIEW

Tape Function Manager Modules

The tape function manager component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QTADSPY–Display Tape Volume (DSPTAP)[1]: This module displays the tape volume and file labels.

-->QTAOPEN–Open Tape File: This module opens a tape file for input or output processing.

> QTAVOPEN–Open Tape Volume: This module opens a tape volume, for input or output processing, at file opening or end-of-volume.
>
> QTAOPCHK–File Open Parameter Checking: This module performs parameter checking on an open file.

-->QTAGET–Tape Get: This module retrieves one or more records for the user.

-->QTAPUT–Tape Put: This module writes one or more records of the user data to the tape.

-->QTACLOSE–Close Tape File: This module closes a tape file to input or output processing.

> QTAVCLOS–Close Tape Volume: This module closes a tape volume, to input or output processing, at file closing or end-of-volume.
>
> QTAEOV–Tape End-of-Volume Processor: This module closes the current volume when an end-of-volume condition is detected and then opens the next volume for a multivolume operation.

[1]This module is a CPP (command processing program).

-->QTAFEOD–Force End of Data: This module positions to the end-of-file for an input file and signals the end-of-file. For an output file, it writes buffered records onto the tape.

-->QTAFEOV–Force End-of-Volume: This module forces an end-of-volume condition which causes the current volume to be closed and a continuation volume, if any, to be opened.

-->QTAINIT–Initialize Tape Volume (INZTAP)[1]: This module initializes a tape volume by writing a volume label and/or tape marks at the beginning of the reel.

-->QTALUDIN–Tape LUD Initialization: This module performs the tape-specific processing required to vary on a tape device description. It initializes the tape part of the LUD-associated space.

-->QTADUMP–Dump Tape Volume (DMPTAP)[1]: This module dumps tape labels and file data to the printer.

QTADMPIO–Dump Tape I/O Processor: This module reads data from the tape, formats it, and writes the data to the printer.

QTAERR–Tape Error Handler: This module performs all message handling (send and receive message functions) for the tape support component. It also handles unexpected I/O feedback responses by signaling an appropriate escape message.

QTAERRIN–Error Table Initialization: This module is called to initialize tables in the associated space of QTAERR.

-->QCHKTAP–Check Tape (CHKTAP)[1]: This module checks mounted volumes for a specific tape volume and/or a specific file.

Tape Operation

Figure TA-1 and the following text describe a tape operation.

1 A high-level language program, system utility, or save/restore component, through the QDMCOPEN module of common data management, calls QTAOPEN to open a tape file for input or output processing.

A An argument list is passed that contains a pointer to the UFCB (user file control block).

The tape to be used is selected by the values in the volume/reels parameters specified by the caller of common data management. QTAOPCHK is called to perform parameter checking.

QTAVOPEN is called and the volume label identifier field is verified if the caller specified a volume ID and if the tape contains standard labels.

B A message is sent to the system operator console if the volume identifier cannot be found. The operator can load another tape and retry the operation or the job can be canceled.

If the file is being opened for input or output EXTEND (*YES):

- The tape file labels are searched for a match to the file sequence number and optionally, to the label name specified by the caller.

  If the file cannot be found, the operation is terminated, and an escape message is signaled.

- The tape is positioned at either the beginning of data for files to be read forward, or at the end of data for files to be read backward.

[1]This module is a CPP (command processing program).

If the file is being opened for output EXTEND (*NO):

- The file SEQNBR indicates where the new file starts.
- Expired files on tape are written over on output. A file is considered to be expired if the file expiration data (in the file HDR1 label) is less than or equal to the system date.
- For standard label files (HDR1, HDR2) header labels are written at the beginning of the file.

Two buffers are used by the tape function manager. Each buffer holds a multiple of tape blocks up to a sum of approximately 8 K bytes if the block size is less than 8 K bytes. Otherwise, each buffer is the same size as the block length.

For fixed-length records, where the block size is less than 8192 bytes, as many blocks as will fit into 8192 and still number 100 blocks or less for an output file, or 256 blocks or less for an input file, constitutes a full buffer. Where the block size is greater than 8192 or for variable-length records of any size, one block constitutes a full buffer.

2 Control is returned to QTAOPEN. If the file has been opened for output, information is written to the file by calling QTAPUT.

A An argument list is passed that contains pointers to the UFCB, an option list, and control information.

B The option list is checked for a put wait operation, and the control list is used (for variable-length files only) to access the record length. If no control list is specified for variable files, maximum record length is assumed.

C Request I/Os are issued to the tape I/O manager when the user has sent the tape function manager enough records to write a buffer full of data.

If the end-of-tape is encountered, QTAEOV is called to switch output volumes.

Note: The save/restore component does not use this interface. Save/restore issues special request I/Os to put data to a tape. See *Save/Restore.*

3 If the file has been opened for input, information can be retrieved from the file by calling QTAGET.

A An argument list is passed that contains pointers to the UFCB, an option list, and control information.

B The option list is checked, and only get next wait is allowed for forward-read files or get previous wait for backward-read files; other requests result in an error message being sent to the caller. The control information is ignored.

Request I/Os are issued to the tape I/O manager when buffers become empty.

If there is no more data on the volume, QTAEOV is called to switch input volumes or signal end-of-file.

Note: The save/restore component does not use this interface. Save/restore issues special request I/Os to retrieve data from a tape. See *Save/Restore*.

4 After a file has been processed, it is closed by calling QTACLOSE through QDMCLOSE.

A An argument list is passed that contains pointers to the ODP (open data path), an index to the device being closed, and the type of close to perform (permanent or temporary).

QTAVCLOS is called and if the file being closed had been opened as an output file:

- The data remaining in the buffers is written to the tape.
- For standard labeled tape an EOF1, EOF2, and double tape marks are written. Nonlabeled tape will have two tape marks written.

If the file being closed had been opened as an input file:

- The tape I/O manager is instructed, by a MODLUD instruction to suspend, deactivate, and reactivate the LUD (logical unit description) session state, to stop processing any current or pending request I/Os.

- If the reel is positioned at the tape mark and all blocks on the reel were processed prior to close, the count of blocks read is compared to the trailer label block count for an *SL file. If a block is missing/extra, an escape message is sent.

If a permanent close is requested, objects created by the tape function manager are destroyed.

5 When a forced end of data is requested, QTAFEOD is called and the following occurs:

A An argument list is passed that points to the UFCB.

B QTAEOV is repeatedly called to switch volumes until an end-of-file exception is signaled to the user. (For multivolume files, the end-of-file exception is signaled after the last volume of the file has been located.)

If the file is opened for output, buffered data is written to tape. QTAEOV is called when the end-of-tape is encountered, to switch output volumes.

6 When a forced end-of-volume is requested, QTAFEOV is called and the following occurs:

A An argument list is passed that points to the UFCB.

B QTAEOV is called to switch volumes. If the file is opened for input, the file is spaced to the end of the current volume. If end-of-file is detected, it is signaled to the user and if end-of-volume, the next volume is opened for input. Subsequent gets retrieve data from this new volume.

If the file is opened for output, all buffered I/O is written to the current volume, ending tape marks (and labels) are written, and next volume is opened. Subsequent puts output data to this new volume.

7 End-of-volume switching occurs automatically within the tape function manager when it is processing a multivolume file.

If a file is open for output, QTAEOV:

- Calls QTAVCLOS to write an EOV1 and EOV2 labels.

- Calls QTAVOPEN to request the mounting of the next volume. QTAVOPEN performs the checks as described for output files. The HDR1 label also contains a volume sequence number that will be one higher than the number written in the previous volume of the file.

If the file is open for input, QTAEOV:

- Calls QTAVCLOS to finish processing of the previous volume; QTAVCLOS checks the trailer label for end-of-volume or end-of-file.

- Calls QTAVOPEN to request the mounting of the next volume if the volume closed was not the final one. The labels are checked to verify that the next volume of the file is in proper sequence.

QTAEOV causes an exception to be signaled to its caller. Either an end-of-volume notify message or an end-of-file status message is sent to the using program.

C Request I/Os communicate to the tape I/O manager the desired action. The tape I/O manager indicates its success or failure in performing the request by returning a message in the machine interface response queue.

QTAERR sends operator messages and program exceptions. It is also called to analyze I/O errors to determine what recovery action is to be performed.

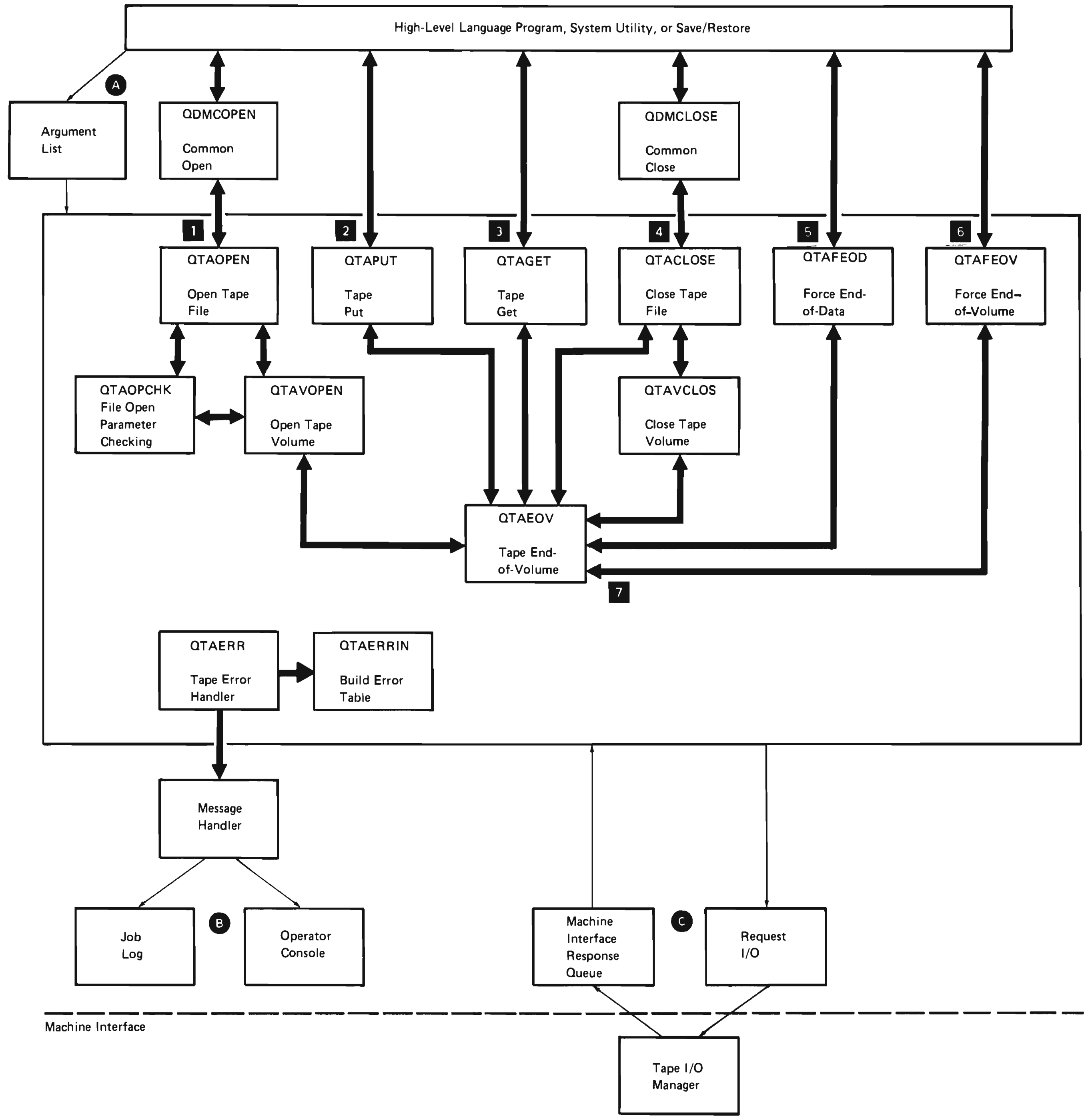

High-Level Language Program, System Utility, or Save/Restore
A
Argument List
QDMCOPEN Common Open
QDMCLOSE Common Close
1
QTAOPEN Open Tape File
2
QTAPUT Tape Put
3
QTAGET Tape Get
4
QTACLOSE Close Tape File
5
QTAFEOD Force End-of-Data
6
QTAFEOV Force End-of-Volume
QTAOPCHK File Open Parameter Checking
QTAVOPEN Open Tape Volume
QTAVCLOS Close Tape Volume
QTAEOV Tape End-of-Volume
7
QTAERR Tape Error Handler
QTAERRIN Build Error Table
Message Handler
Job Log
B
Operator Console
Machine Interface Response Queue
C
Request I/O
Machine Interface
Tape I/O Manager

Figure TA-1. Tape Operation Overview

INTRODUCTION

The testing component of the CPF (control program facility) is used to test and debug programs under a test environment that is similar to the actual production environment. Special statements do not have to be inserted into the program. The test component provides the special tools needed to debug and test a program.

When testing is requested, a special environment called debug mode is created. This mode prevents unintentional modification to data base files in the production libraries. Only data base files in test libraries can be updated. To establish a debug mode, an Enter Debug (ENTDBG) command is issued. Once in a debug mode, the testing commands can be used to test and debug programs. The debug mode is ended by the End Debug (ENDDBG) command.

GENERAL OVERVIEW

The testing component consists mainly of CPP (command processing program) modules. These CPPs let the user:

- Add or remove programs from debug mode
- Add or remove breakpoints
- Add or remove traces
- Change program variables and pointers
- Change the Enter Debug command parameters
- Display debug, breakpoint, program variable, and trace information
- Resume program execution at a breakpoint
- Cancel a request that was entered at an earlier request level.

In addition to these CPPs, there are also support and event handler modules in the Testing component.

Testing Modules

The testing component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

Command Processing Modules

-->QTECADBP–Add Breakpoint (ADDBKP)[1]: This module is used to establish one or more breakpoints in a program that is being debugged.

-->QTECADPG–Add Program (ADDPGM)[1]: This module prepares debugging operations for the specified programs.

-->QTECADTR–Add Trace (ADDTRC)[1]: This module processes a request to establish a trace operation in a program that is being debugged.

-->QTECCHDB–Change Debug (CHGDBG)[1]: This module is used to change debug options for the current debug session.

-->QTECCHHP–Change HLL Pointer (CHGHLLPTR)[1]: This module changes the addressability of HLL pointers.

-->QTECCHPT–Change Pointer (CHGPTR)[1]: This module changes the addressability and type of system or space pointers.

-->QTECCHVR–Change Program Variable (CHGPGMVAR)[1]: This module changes the value of a program data object.

-->QTECCLTD–Clear Trace Data (CLRTRCDTA)[1]: This module causes the trace data area to be cleared.

[1]This module is a CPP (command processing program).

-->QTECCNRQ–Cancel Request (CNLRQS)[1]: This module cancels a previous request. It causes termination and deletion of all invocations associated with the request.

-->QTECDSBP–Display Breakpoint (DSPBKP)[1]: This module displays to the user which breakpoints are set in the programs being debugged, and the variables to be displayed at those breakpoints.

-->QTECDSDB–Display Debug (DSPDBG)[1]: This module displays to the user the current status of the debug session.

-->QTECDSTD–Display Trace Data (DSPTRCDTA)[1]: This module displays trace data to the user.

-->QTECDSTR–Display Trace (DSPTRC)[1]: This module displays to the user the trace ranges that have been set and the variables that have been specified for those ranges.

-->QTECDSVR–Display Program Variables (DSPPGMVAR)[1]: This module displays a maximum of ten specified variables from the program that is being debugged.

-->QTECENDB–End Debug (ENDDBG)[1]: This module is used to terminate the debug mode.

-->QTECNTDB–Enter Debug Mode (ENTDBG)[1]: This module is used to establish the debug mode within a process.

-->QTECRMBP–Remove Breakpoint (RMVBKP)[1]: This module is used to remove breakpoints in a program that is being debugged.

-->QTECRMPG–Remove Program (RMVPGM)[1]: This module is used to remove programs from the debug mode.

-->QTECRMTR–Remove Trace (RMVTRC)[1]: This module is used to remove all traces from a program or to remove just the trace ranges that were specified in a program being traced.

-->QTECRSBP–Resume Program Execution at a Breakpoint (RSMBKP)[1]: This module causes a program that is stopped at a breakpoint to resume execution.

Support and Event Handler Modules

The following modules provide support and handle events for the CPPs:

QTEMBKCU–Breakpoint Cleanup: This module is invoked if either QTEVIREF or QTEMDEBP is terminated in an abnormal manner. The module adjusts the breakpoint count and destroys any associated created space.

QTEMFMT–Format Data: This module converts the attributes and data to displayable characters. This module can be called by QTECDSVR and QTEVIREF.

QTEMGTHL–Get High-Level Language Statement Identifier: This module gets the high-level language statement identifier associated with a given machine interface instruction number.

QTEMGTVR–Get Program Variables: This module gets the attributes and values of the program variables. QTEMGTVR can be called by QTECDSVR, or QTEVIREF.

QTEMLOBJ–Locate Program Objects: This module determines the address of variables within the program. It can be called by QTECCHHP, QTECCHPT, QTECCHVR, and QTEMGTVR.

QTEMVFOR–Verify ODV References: This module performs checks in the ODV (object directory vector) for specified variables. It can be called by QTECADBP, QTECADTR, QTECCHHP, QTECCHPT, QTECCHVR, and QTECDSVR.

-->QTEVIREF–Instruction Reference Event Handler: This module handles the instruction reference event when it is signaled. If the instruction passed to it is at a breakpoint, this module formats and displays the instruction number and variable data. If the instruction passed to it is being traced, this module formats and stores the instruction number and variable data.

[1]This module is a CPP (command processing program).

-->QTEMDEBP–Default Exception Breakpoint Module: This module is invoked if an unmonitored escape message occurs in an interactive debug job. The module displays the message and provides optional command entry capability similar to a normal, user-defined breakpoint.

The following module is called during a cancel request operation:

QTEMRCLM–Reclaim Resources: This module is invoked during execution of a Cancel Request command. It invokes the reclaim files and reclaim PSSA (process static storage area) functions for the invocations that have been eliminated during the request cancelation.

The symbol table entry for any variable can contain a format segment to identify a program that is provided by the HLL to process the variable. Modules QTEMVFOR, QTEMGTVR, QTEMLOBJ, QTECCHVR, QTECCHPT, and QTECCHHP all can call a format program to perform such functions as locating the variable or formatting its current value for display.

Master Debug Communication Object

The MDCO (master debug communication object) is one of the principle control blocks used by the testing component. There is one MDCO for each debug job. It is created by QTECNTDB and destroyed by QTECENDB. The MDCO contains information about the debug session. This information includes:

- Values of the various debug job parameters
- Names of all the programs currently being debugged
- Pointers to the DCO (debug communication object) for programs currently being debugged
- Pointers to other objects, such as trace data spaces, PSSA (process static storage area) header, and PASA (process automatic storage area) header.
- When a breakpoint occurs, information about it is put into the MDCO. That information includes:
  - Program name
  - Invocation level
  - Breakpoint instruction number

Debug Communication Object

The DCO is the other principal object used by the testing component. There is one DCO for each program currently being debugged. They are created by QTECADPG and destroyed by QTECRMPG. The DCO contains:

- A system pointer to the program being debugged
- A system pointer to a space for the program template materialization
- Space pointers for basing based views of program template components
- Pertinent count and length fields and indicators

Enter Debug Command

Figure TE-1 and the following text describe an Enter Debug (ENTDBG) command operation.

1 The command analyzer decodes an Enter Debug command and control is transferred to QTECNTDB. QTECNTDB then creates space for the MDCO and puts the addressability to it in a pointer located in the job's WCB (work control block). The appropriate MDCO fields are also updated.

2 QTECNTDB establishes an event monitor for the instruction reference event, identifying QTEVIREF as the event handler program.

3 QTECNTDB gets addressability to the WCB, finds the job type (interactive or batch) to set the indicator in the MDCO, and then sets the debug mode indicator on in the WCB.

4 If the PGM parameter is specified in the Enter Debug command, QTECADP is called to put the program in debug mode. Prior to calling QTECADPG, a function check monitor is enabled. If a function check occurs, everything that has occurred is undone (the event monitor is canceled, the WCB is changed to its original format, the MDCO is destroyed) and the function check is resignaled.

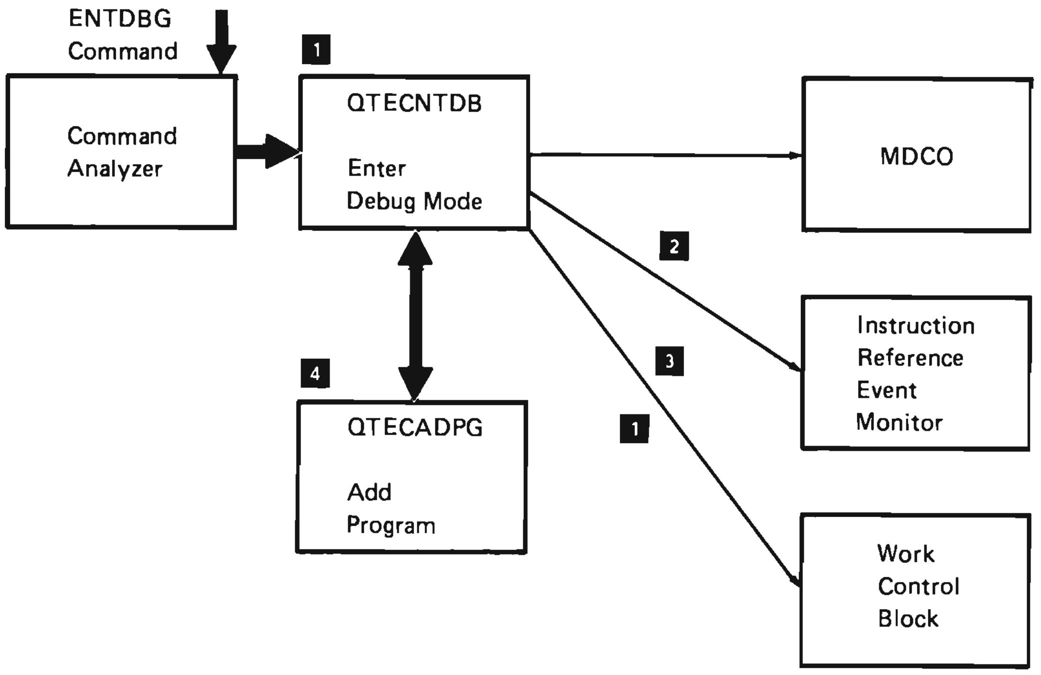


Figure TE-1. Enter Debug Command Overview

Add Program Command

Figure TE-2 and the following text describe an Add Program (ADDPGM) command operation.

1 The command analyzer or QTECENTB calls QTECADPG. QTECADPG locates the MDCO. It checks each name in the PGM parameter list for validity, program found, not already under debug, program observable, and so forth. If all checks are okay, QTECADPG proceeds.

2 A DCO space is created for each program.

3 A program template materialization space (QTESPTSP) is created for each program, and the programs are materialized.

4 The appropriate MDCO fields are updated.

5 The appropriate DCO fields are updated.

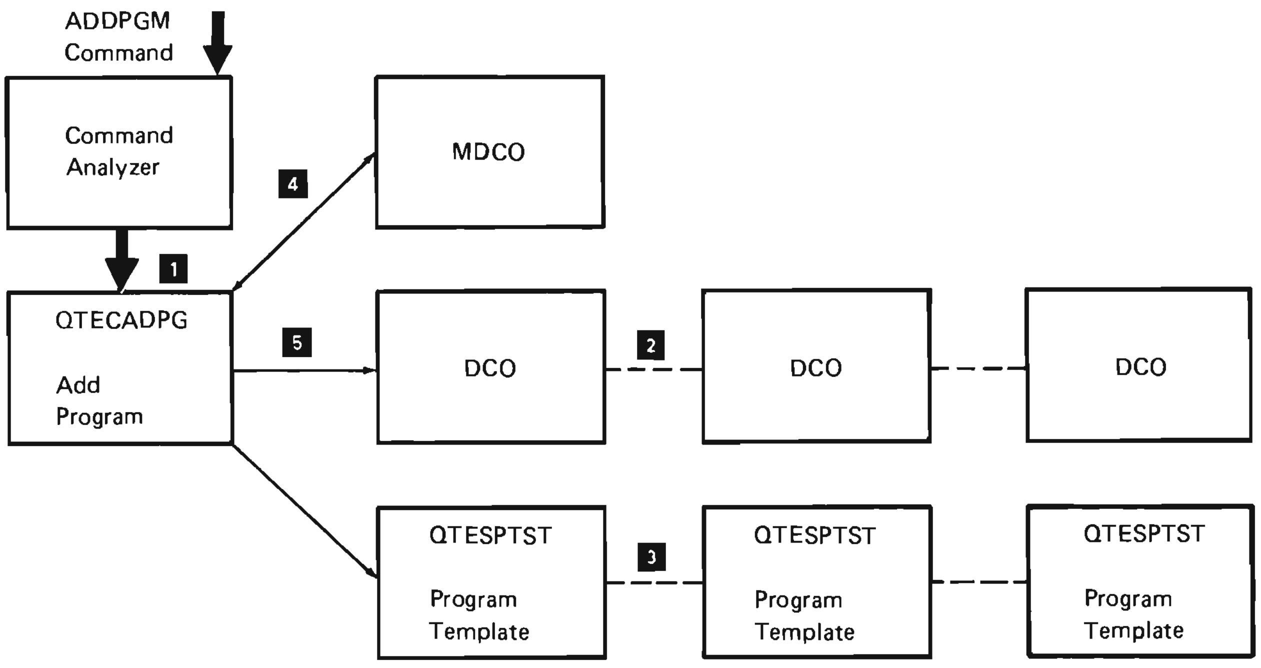

ADDPGM Command
Command Analyzer
1
QTECADPG Add Program
4
MDCO
5
DCO
2
DCO
DCO
QTESPTST Program Template
3
QTESPTST Program Template
QTESPTST Program Template

Figure TE-2. Add Program Command Overview

Remove Program Command

Figure TE-3 and the following text describe a Remove Program (RMVPGM) command operation.

1 The command analyzer decodes a Remove Program command and control is transferred to QTECRMPG or QTECRMPG is called by QTECENDB. QTECRMPG locates the MDCO and checks all of the names in the PGM parameter for validity.

2 QTECRMPG gets addressability to the DCO for each program to be removed.

3 From the DCO of each program to be removed, QTECRMPG gets addressability to and then destroys each created space used by the other debug CPPs.

4 QTECRMPG destroys each DCO and updates pertinent fields in the MDCO, nulls pointers to the DCOs, nulls name fields, and decrements the count of programs in the debug mode.

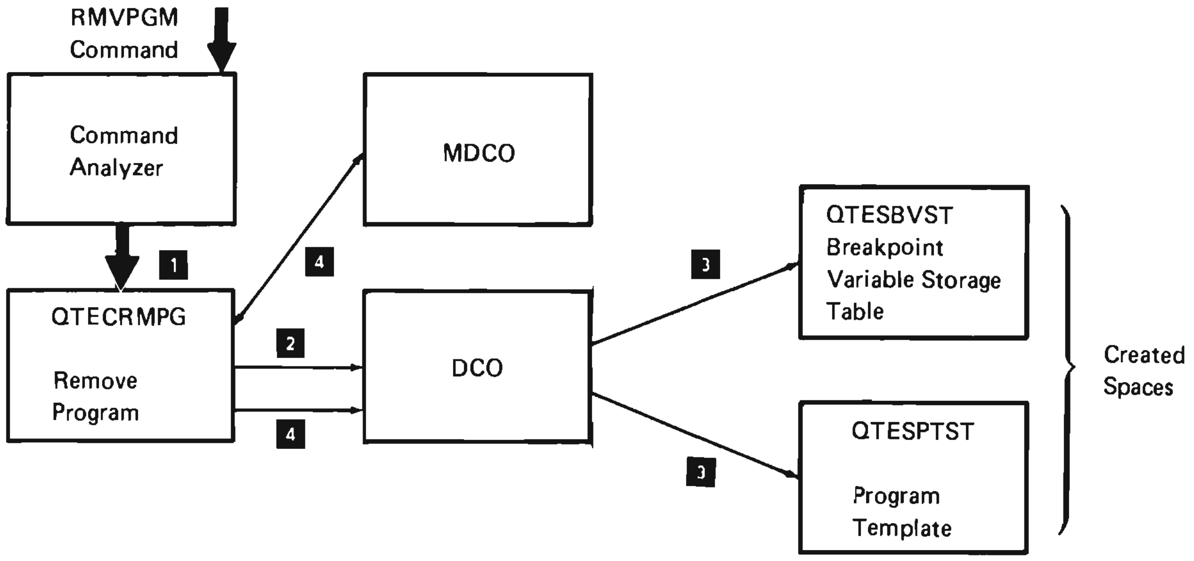


Figure TE-3. Remove Program Command Overview

End Debug Command

Figure TE-4 and the following text describe an End Debug (ENDDBG) command operation.

1 The command analyzer decodes an End Debug command and control is transferred to QTECENDB. QTECENDB locates the WCB and changes the mode bits to normal.

2 QTECENDB cancels the event monitor for the instruction reference event.

3 The MDCO is located and the count of programs in debug mode is checked.

4 If the program count is not zero, QTECRMPG is called to perform a cleanup operation.

5 If there are trace data spaces, QTECCLTD is called to destroy them.

6 The MDCO is destroyed, and the pointer to it in the WCB is nulled.

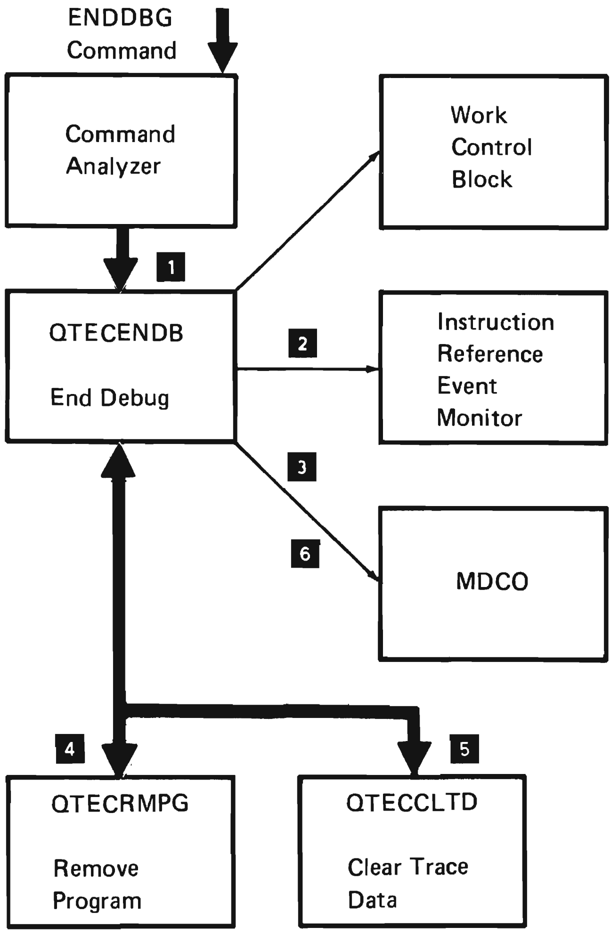


Figure TE-4. End Debug Command Overview

Add Breakpoint Command

Figure TE-5 and the following text describe an Add Breakpoint (ADDBKP) command operation.

1 The command analyzer decodes an Add Breakpoint command and control is transferred to QTECADBP. QTECADBP locates the MDCO and uses the PGM parameter value to locate the DCO.

2 From information in the DCO, QTECADBP determines if breakpoint table spaces exist.

3 If spaces do not exist, they are created in the breakpoint definition table for instructions where breakpoints will be added, and in QTESBVST for variables to be displayed at breakpoints.

4 QTEMVFOR is called to validity check the input from the PGMVAR parameter. If any high-level language names are used, the BOM (break offset mapping) and the symbol table of the program template are used in the validity checking operation.

5 If the validity check is okay, instruction data is added to the breakpoint definition table in the DCO and variable data is added to QTESBVST. These tables are used by QTEVIREF when an instruction reference event is signaled.

6 The instructions to be traced are sent to machine observation support.

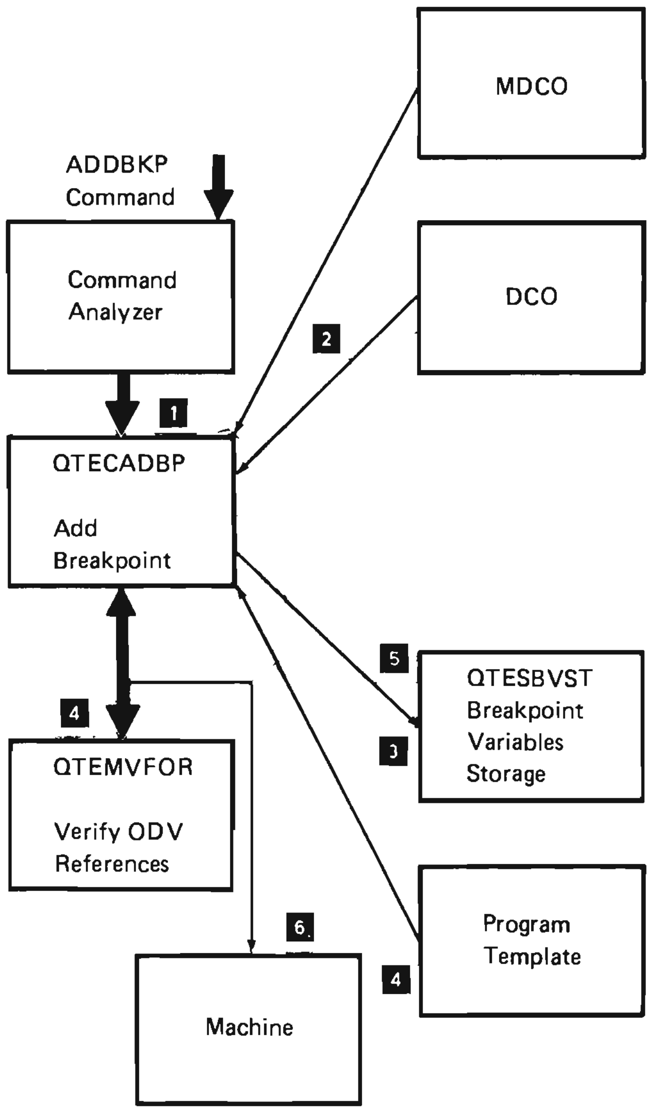


Figure TE-5. Add Breakpoint Command Overview

Remove Breakpoint Command

Figure TE-6 and the following text describe a Remove Breakpoint (RMVBKP) command operation.

1 The command analyzer decodes a Remove Breakpoint command and control is transferred to QTECRMBP. QTECRMBP locates the MDCO and the DCOs for the program specified by the PGM parameter or all programs as specified by *ALL.

2 From information in the DCO, QTECRMBP deletes entries from the breakpoint definition table as specified by the STMT parameter.

3 QTECRMBP removes machine traces from all specified instructions.

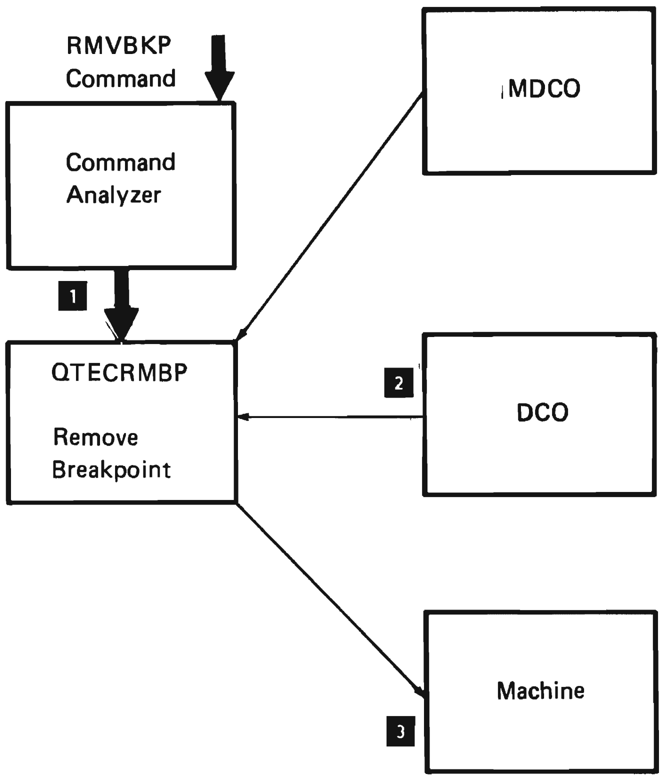


Figure TE-6. Remove Breakpoint Command Overview

Add Trace Command

Figure TE-7 and the following text describe an Add Trace (ADDTRC) command operation.

1 The command analyzer decodes an Add Trace command and control is transferred to QTECADTR. QTECADTR locates the MDCO and the DCO for the specified program.

2 After validity checking all parameter values, a QTESTIO is created to contain information about instructions to be traced and a QTESTDO is created to contain information about variables to be monitored and displayed.

3 If any high-level language names are used, the BOM (break offset mapping) and symbol table of the program template are used in the validity checking operation. QTEMVFOR is also used to validate ODV numbers.

4 If all validity checks are okay, the DCO, QTESTIO, and QTESTDO are updated. The instructions to be traced are then sent to the machine support.

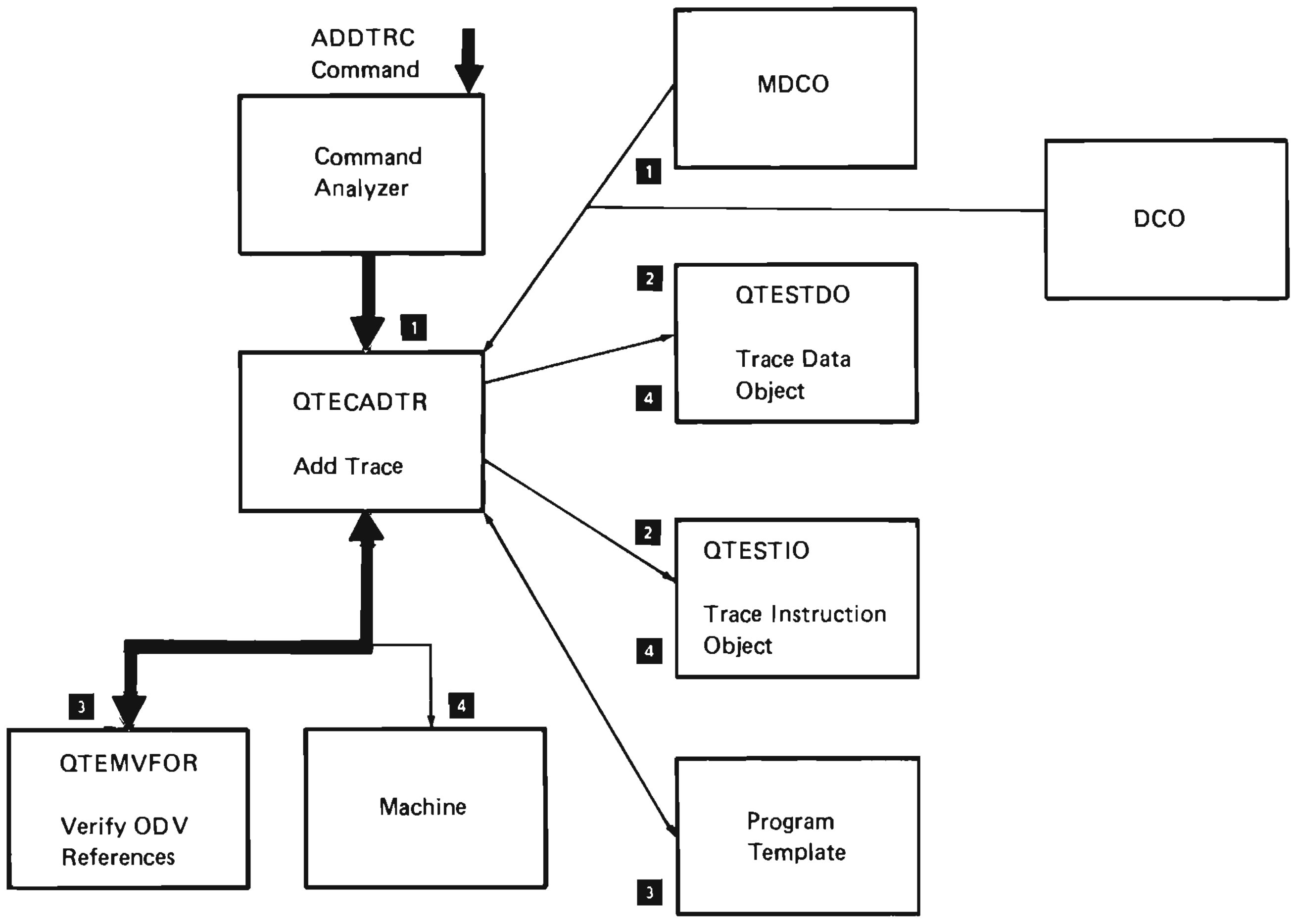


Figure TE-7. Add Trace Command Overview

Remove Trace Command

Figure TE-8 and the following text describe a Remove Trace (RMVTRC) command operation.

1 The command analyzer decodes a Remove Trace command and control is transferred to QTECRMTR. QTECRMTR locates the MDCO and the DCO for the requested program.

2 From information in the DCO, the QTESTIO is located.

3 QTECRMTR searches QTESTIO for the ranges to be removed and when found, deletes them.

4 The breakpoint instruction reference table in the DCO is searched to see if there are any breakpoints for the instructions being removed.

5 QTECRMTR removes the machine support for all instructions that had trace only (no breakpoint) in effect for them.

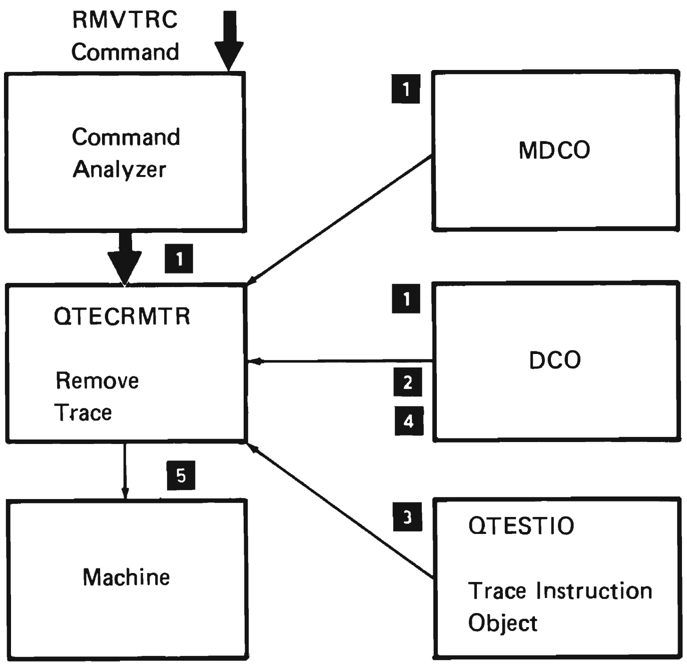


Figure TE-8. Remove Trace Command Overview

Change Debug Command

Figure TE-9 and the following text describe a Change Debug (CHGDBG) command operation.

1 The command analyzer decodes a Change Debug command and control is transferred to QTECCHDB. QTECCHDB locates the MDCO and changes the debug job parameters as specified in the CHGDBG command.

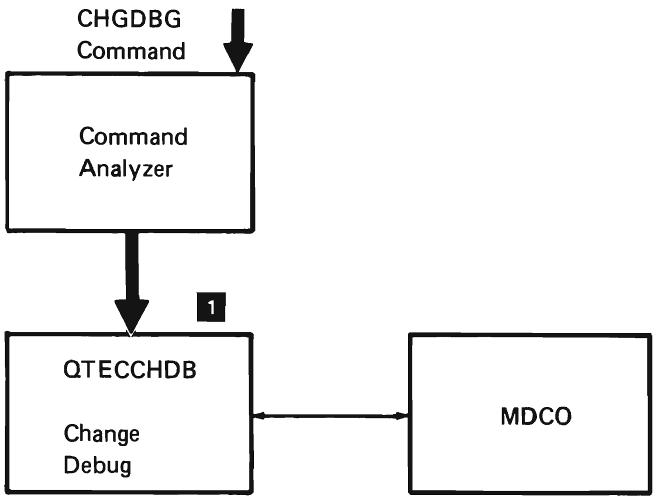


Figure TE-9. Change Debug Command Overview

Display Debug Command

Figure TE-10 and the following text describe a Display Debug (DSPDBG) command operation.

1 The command analyzer decodes a Display Debug command and control is transferred to QTECDSDB. QTECDSDB locates the MDCO, which provides information about the job type and the debug session parameters for the display.

2 QTECDSDB opens the appropriate device file.

3 QTECDSDB uses the PASA (process automatic storage area) chain to gather the information about the invocation stack, including the high-level language statement identifier from QTEMGTHL, and builds a record for each program in the stack.

4 The data to display is sent to the device file. Information about other programs under debug but not invoked is built and sent to the device file and then the device file is closed.

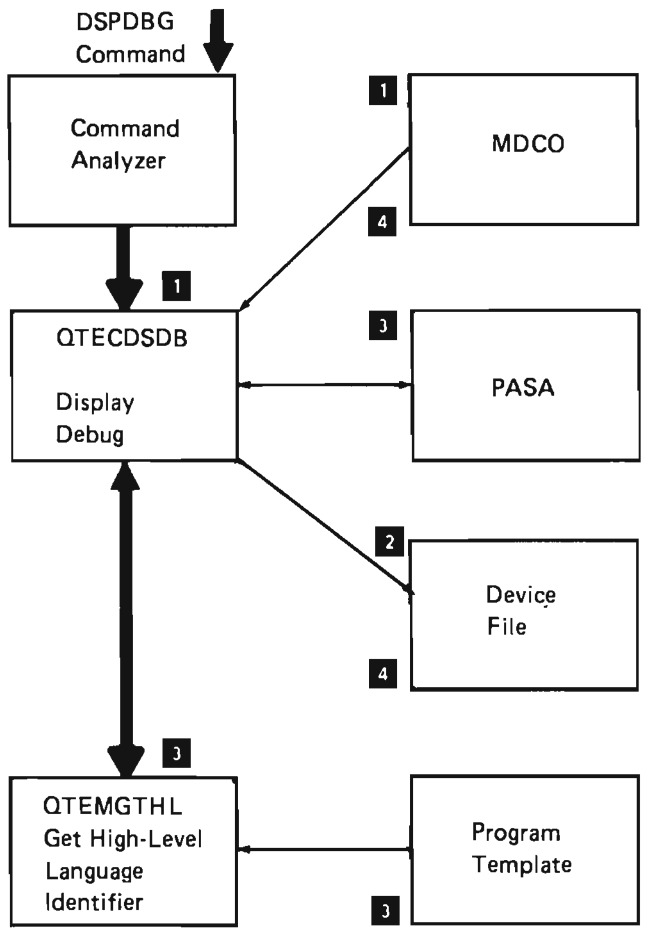


Figure TE-10. Display Debug Command Overview

Display Breakpoint Command

Figure TE-11 and the following text describe a Display Breakpoint (DSPBKP) command operation.

1 The command analyzer decodes a Display Breakpoint command and control is transferred to QTECDSBP. QTECDSBP locates the MDCO and the DCOs as specified by the PGM parameter.

2 The appropriate device file is opened.

3 Using information in the DCO, QTECDSBP processes the breakpoint data for the programs as requested by the command. QTESBVST is accessed and information records are built.

4 The information records are sent to the device file. QTECDSBP checks breakpoint data in the MDCO for program levels stopped at breakpoints and builds and sends those information records to the device file. After all of the information has been sent to the device file, it is closed.

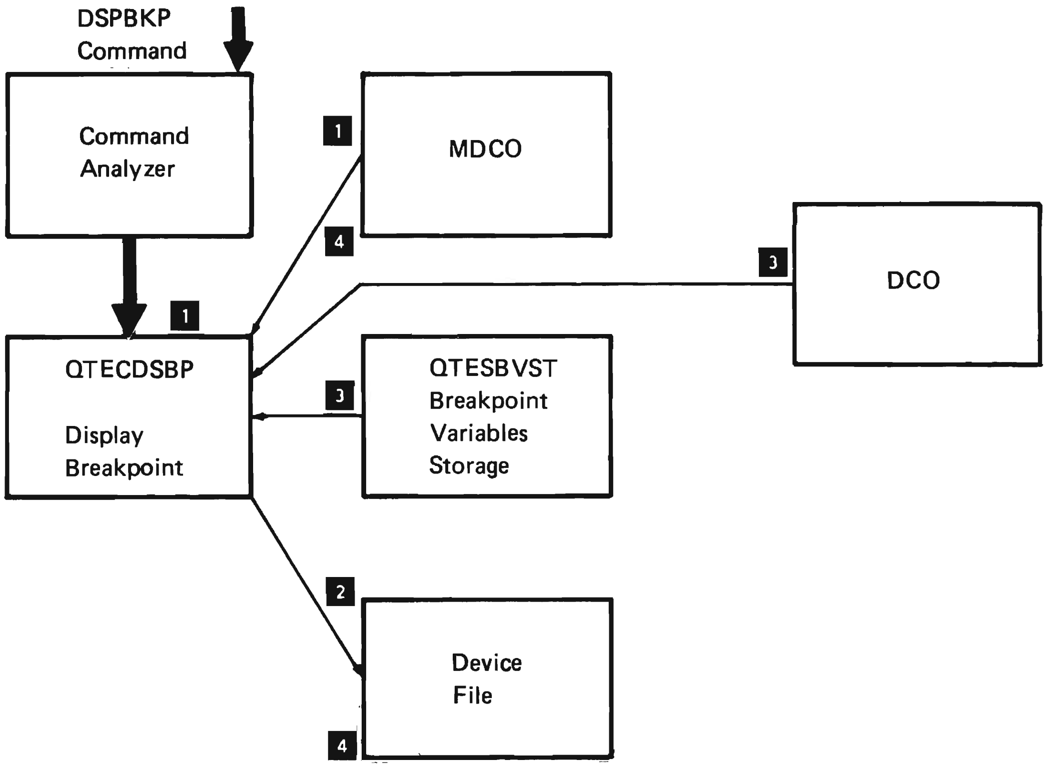

DSPBKP Command
Command Analyzer
1
MDCO
4
3
DCO
1
QTECDSBP Display Breakpoint
3
QTESBVST Breakpoint Variables Storage
2
Device File
4

Figure TE-11. Display Breakpoint Command Overview

Display Trace Command

Figure TE-12 and the following text describe a Display Trace (DSPTRC) command operation.

1 The command analyzer decodes a Display Trace command and control is transferred to QTECDSTR. QTECDSTR locates the MDCO and the DCOs as specified by the PGM parameter.

2 The appropriate device file is opened.

3 From information in DCO, QTESTIO and QTESTDO are located. QTECDSTR then builds records of the traces in effect and the variables to be displayed. This information is sent to the device file. After all records have been sent, the device file is closed.

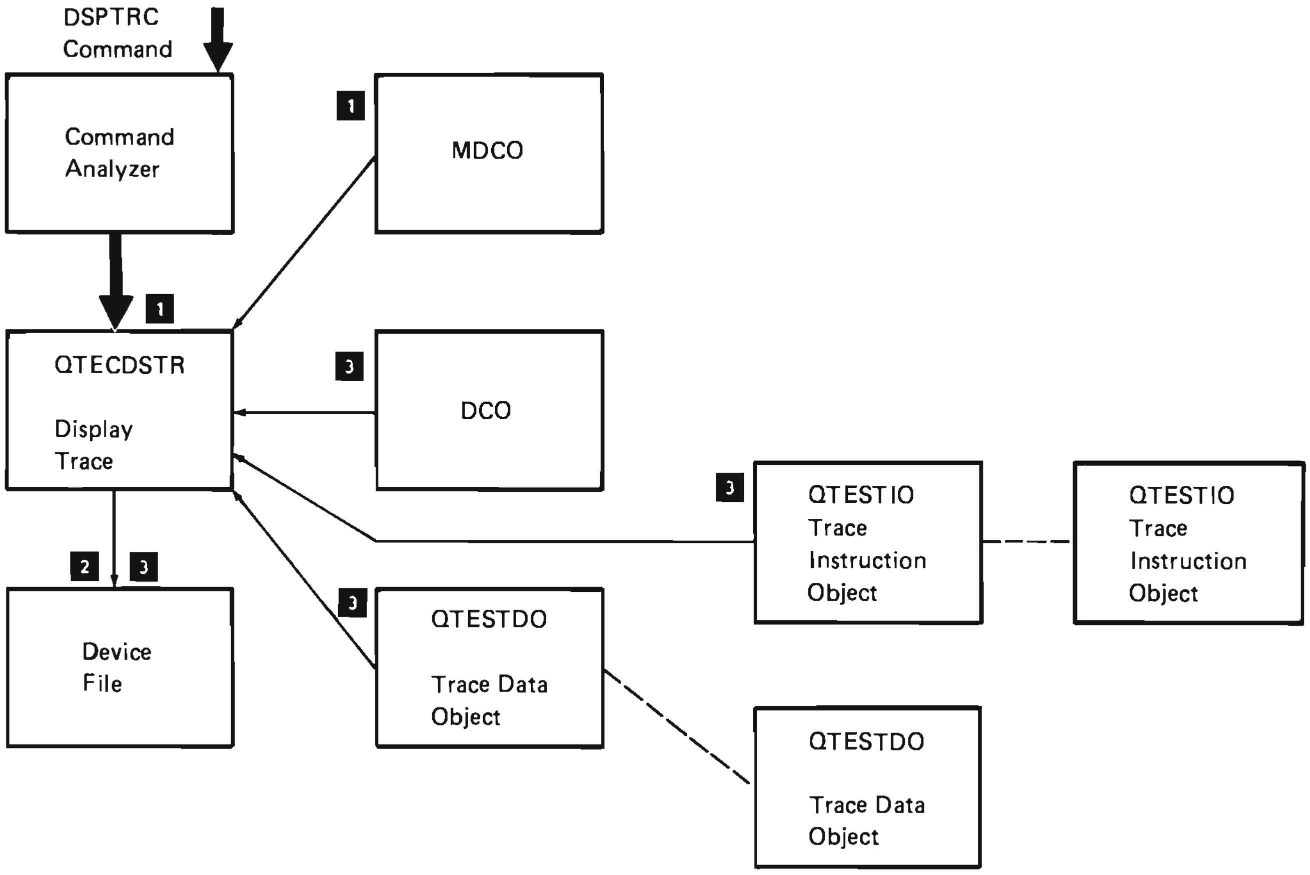

DSPTRC Command
Command Analyzer
1
MDCO
1
QTECDSTR Display Trace
3
DCO
3
QTESTIO Trace Instruction Object
QTESTIO Trace Instruction Object
2
3
Device File
3
QTESTDO Trace Data Object
QTESTDO Trace Data Object

Figure TE-12. Display Trace Command Overview

Display Program Variable Command

Figure TE-13 and the following text describe a Display Program Variable (DSPPGMVAR) command operation.

1 The command analyzer decodes a Display Program Variable command and control is transferred to QTECDSVR. QTECDSVR locates the MDCO and then the DCO for the requested program.

2 QTECDSVR calls QTEMVFOR to validity check the requested variables and verify the ODVs.

3 If all checks are okay, QTEMGTVR is called to get values for all of the variables.

4 QTEMFMT is called to output records for each variable from the information returned by QTEMGTVR. The appropriate device file is opened, the information records sent to it, and then the device file is closed.

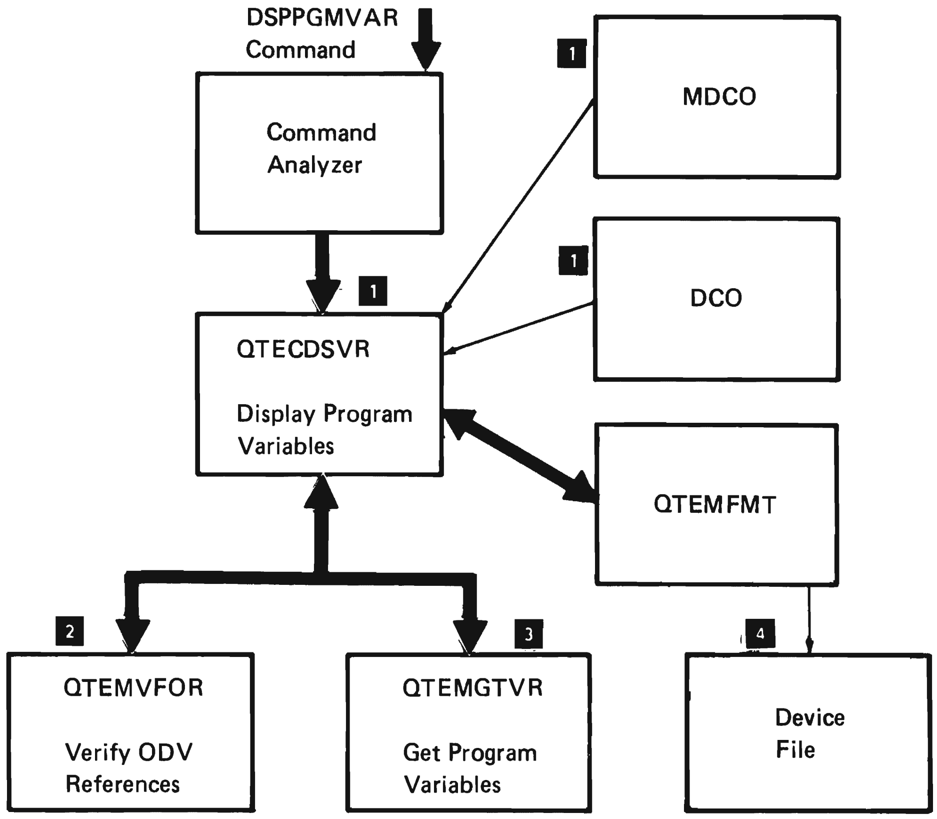


Figure TE-13. Display Program Variable Command Overview

Change Program Variable Command

Figure TE-14 and the following text describe a Change Program Variable (CHGPGMVAR) command operation.

1 The command analyzer decodes a Change Program Variable command and control is transferred to QTECCHVR. QTECCHVR locates the MDCO and then the DCO for the requested program.

2 The variables to be changed are validity checked using QTEMVFOR.

3 QTEMLOBJ is called to get the address of the current instance of the object.

4 QTECCHVR validity checks the values in the VALUE parameter with respect to the attributes of the object and then modifies the variable value.

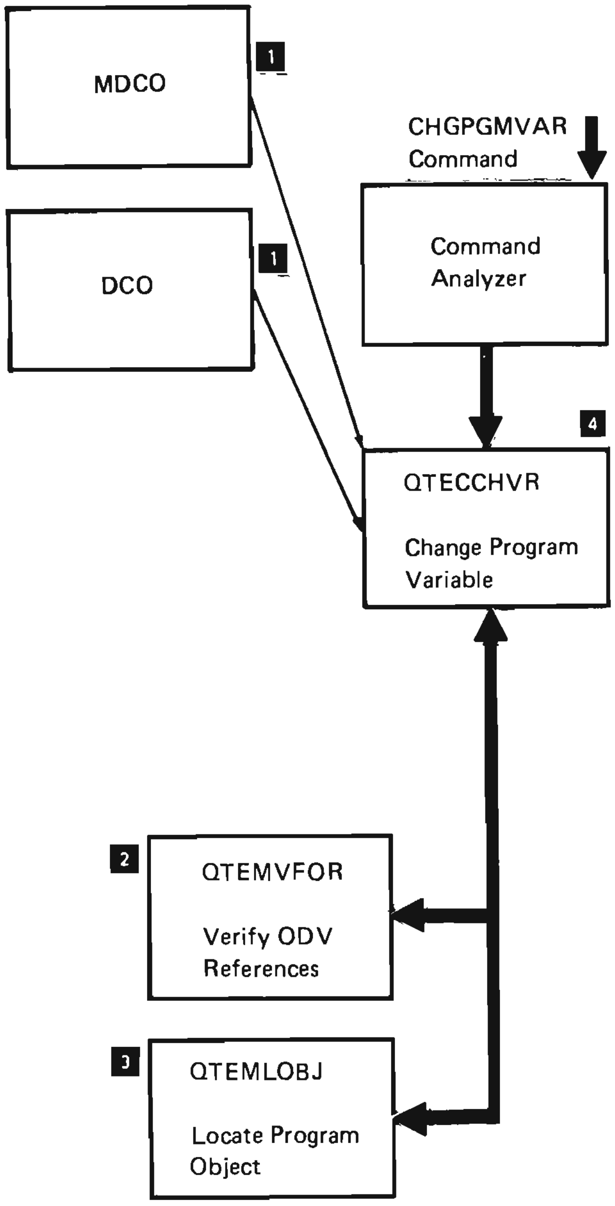


Figure TE-14. Change Program Variable Command Overview

Change Pointer Command

Figure TE-15 and the following text describe a Change Pointer (CHGPTR) command operation.

1 The command analyzer decodes a Change Pointer command and control is transferred to QTECCHPT. QTECCHPT locates the MDCO and then the DCO for the requested program.

2 The pointer variable to be changed is validity checked using QTEMVFOR.

3 QTEMLOBJ is called to get the address of the current instance of the pointer object.

4 The command parameters are validity checked against the pointer type and if there are no errors, the pointer values are modified.

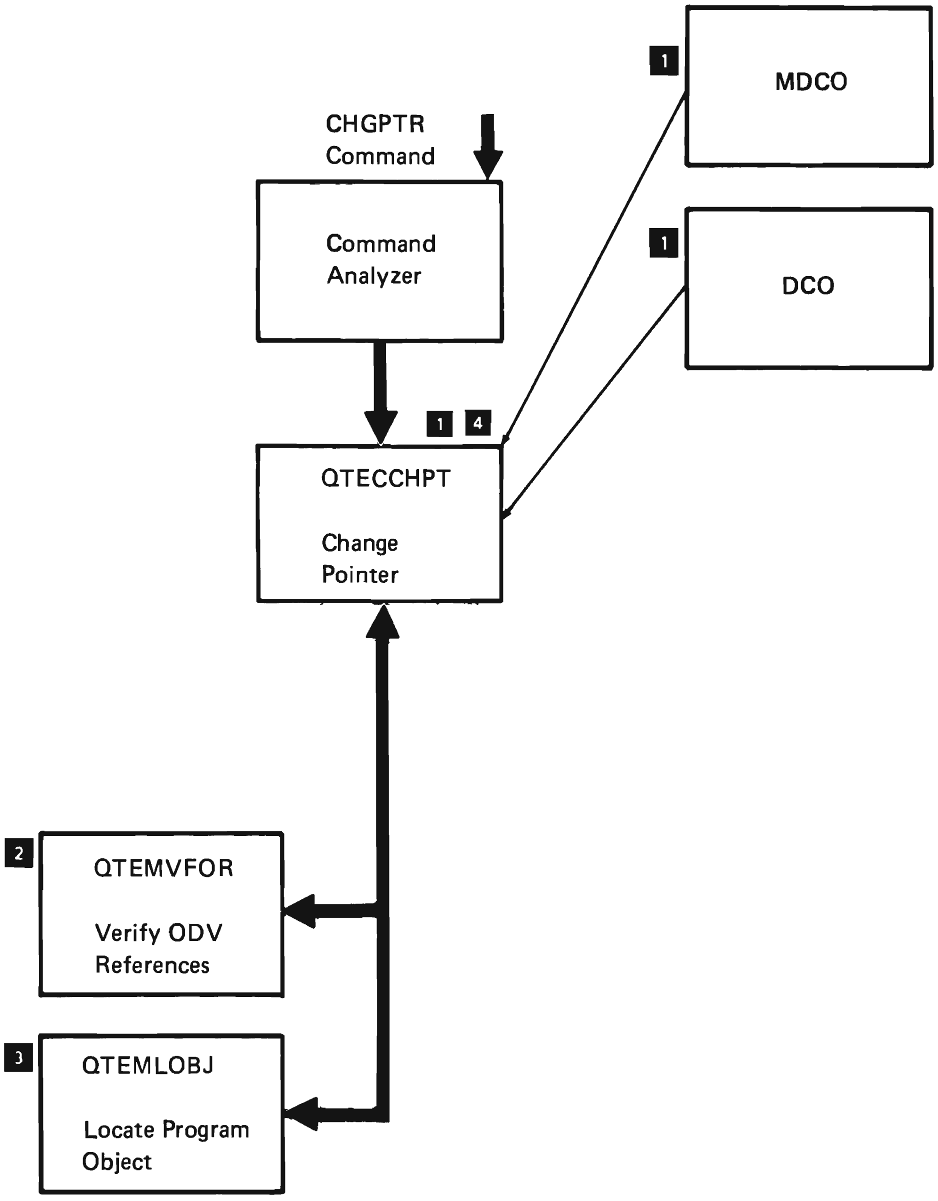


Figure TE-15. Change Pointer Command Overview

Display Trace Data Command

Figure TE-16 and the following text describe a Display Trace Data (DSPTRCDTA) command operation.

1 The command analyzer decodes a Display Trace Data command and control is transferred to QTECDSTD. QTECDSTD locates the MDCO and checks the pointer to the active trace data space. If it is not null, QTECDSTD continues processing.

2 The appropriate device file is opened.

3 QTECDSTD searches the trace data space(s) to the end of the chain and sends trace records to the device file. The device file is then closed.

4 If the command requests that the trace data be cleared, QTECCLTD is called.

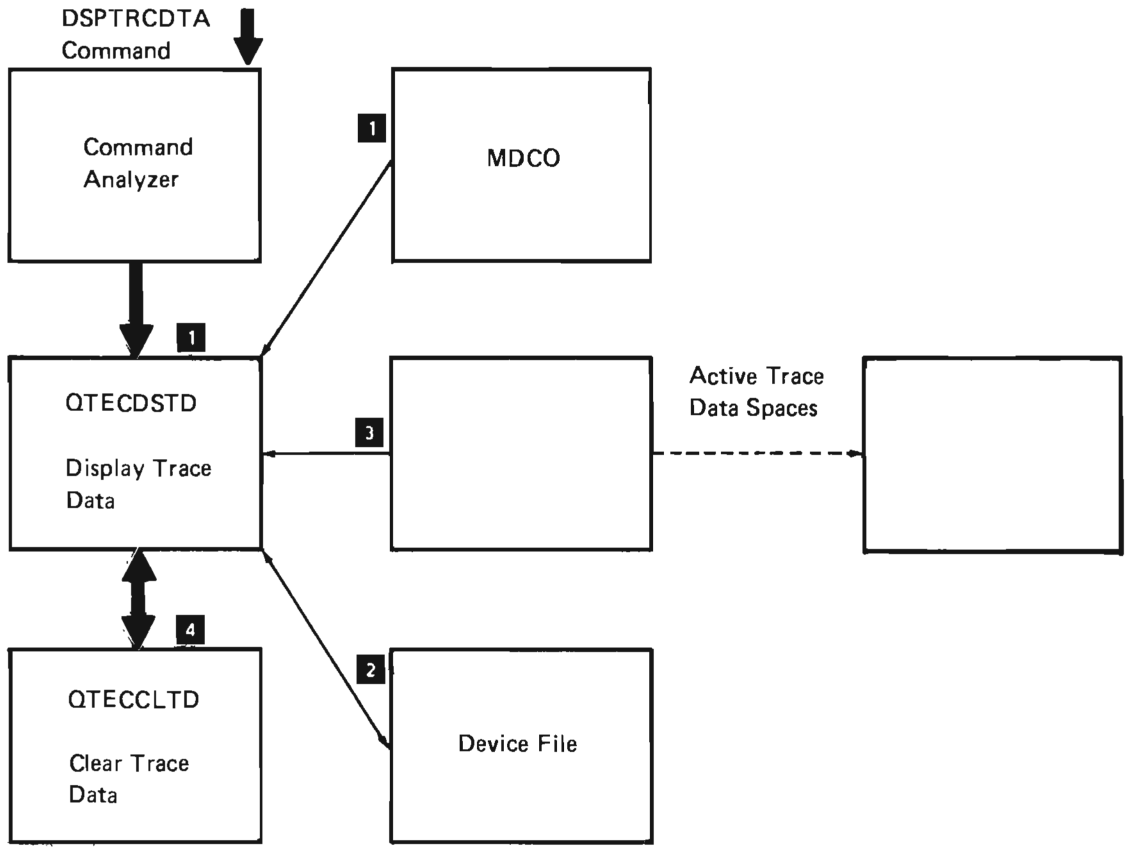


Figure TE-16. Display Trace Data Command Overview

Clear Trace Data Command

Figure TE-17 and the following text describe a Clear Trace Data (CLRTRCDTA) command operation.

1 The command analyzer decodes a Clear Trace Data command and control is transferred to QTECCLTD. QTECDSTD or QTECENDB can also call QTECCLTD. QTECCLTD locates the MDCO and checks the pointers to the active and available trace data chains. If at least one of the chains is not null, processing proceeds.

2 QTECCLTD proceeds through the active chain, destroying all spaces.

3 All spaces on the available chain are destroyed by QTECCLTD.

4 QTECCLTD updates the MDCO fields, setting the pointers to null and the counters to zero.

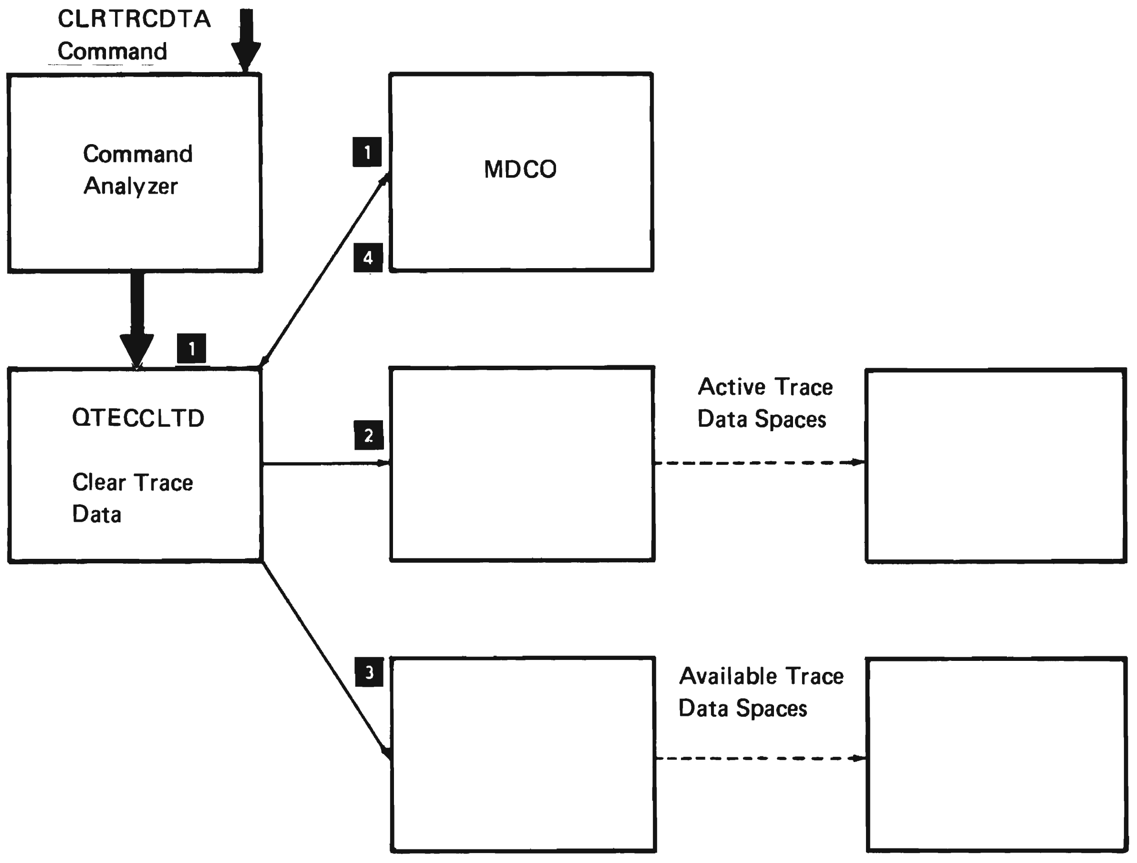


Figure TE-17. Clear Data Command Overview

Resume Breakpoint Command

Figure TE-18 and the following text describe a Resume Breakpoint (RSMBKP) command operation.

1 The command analyzer decodes a Resume Breakpoint command and control is transferred to QTECRSBP. QTECRSBP locates the MDCO and checks to ensure that the program is stopped at a user-defined or trace full breakpoint.

2 An exception is signaled. If the program is at a proper breakpoint, the CPF1901 no error exception is signaled. This exception causes a folding up of the invocation stack. If the program is not at a proper breakpoint, an error exception is signaled.

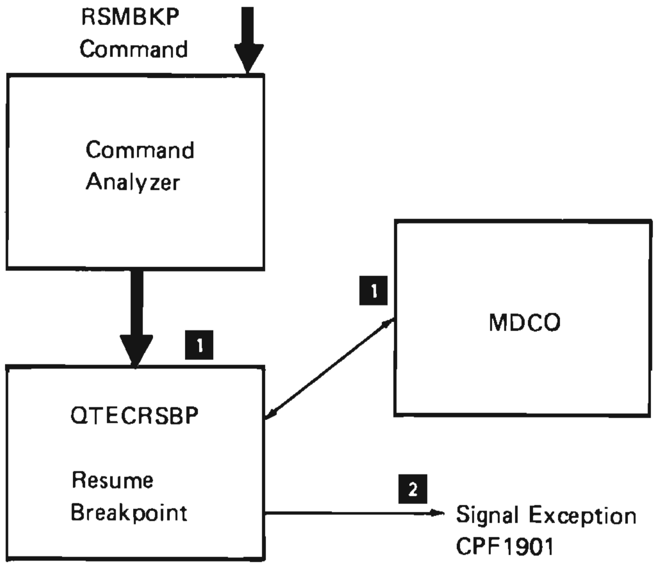


Figure TE-18. Resume Breakpoint Command Overview

Cancel Request Command

Figure TE-19 and the following text describe a Cancel Request (CNLRQS) command operation.

1. The command analyzer decodes a Cancel Request command and control is transferred to QTECCNRQ. QTECCNRQ can also be called by QTEVIREF or QTEMDEBP if CF1 key is selected from a breakpoint display, or an unmonitored message breakpoint display, respectively. QTECCNRQ gets addressability to the head entry of the PASA (process automatic storage area) and from there gets addressability to the invocation entry for the current invocation (QTECCNRQ).

2. QTECCNRQ searches invocation entries in the PASA from its own toward the top of the stack, looking for the request level that is to be canceled. The request level is contained in the invocation control entry of the JMQ (job message queue), which is located by adding the offset value in the PASA invocation entry to the address of the JMQ. The address of the JMQ is obtained from the WCB. While searching, a count of the invocations above this module is maintained. When the proper invocation entry is found, the proper relative invocation to signal to is known.

3. Before performing the cancel function, QTECCNRQ sends CPF1966 (a scope message) to the invocation just below the one to be returned to and passes a system pointer to QTEMRCLM. This tells the system to invoke QTEMRCLM, when the invocation to which the scope message was sent, is destroyed. QTEMRCLM issues instructions to close files and reclaim static storage associated with destroyed invocations.

4. CPF1907, not an error exception, is signaled by QTECCNRQ to the proper request processor program invocation.

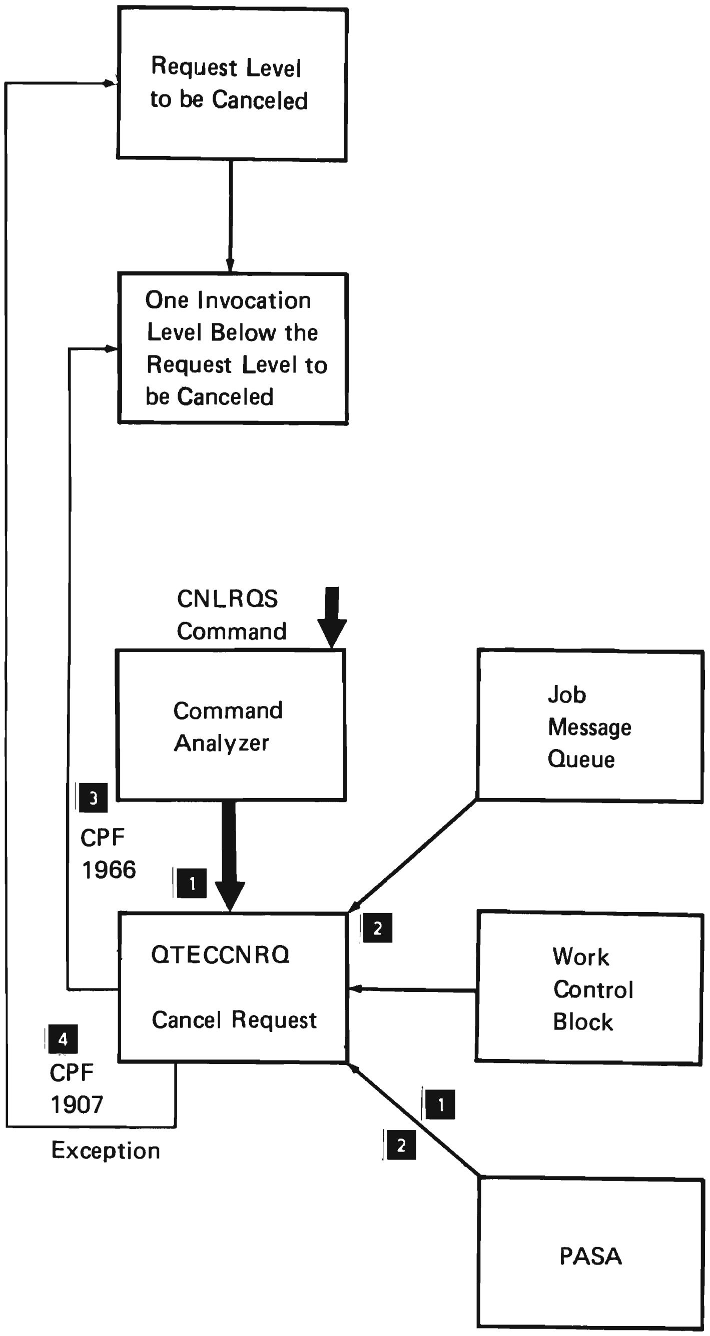


Figure TE-19. Cancel Request Command Overview

Instruction Reference Event Handling

Figure TE-20 and the following text describe the operation of the instruction reference event handler.

1 QTEVIREF is invoked by machine event management when an instruction reference event occurs.

2 Addressability to the MDCO is obtained.

3 The machine event data, including a pointer to the program from which the event was signaled, is obtained.

4 QTEVIREF uses the program pointer to find the DCO for the program that contains the breakpoint or trace point.

5 The breakpoint instruction table in the DCO is searched for the instruction number that was part of the machine event data. If the instruction number is found, all of the following steps (**6** through **8**) occur. If the instruction number is not found, it is assumed to have been a trace point.

6 A request message is sent to and received from the program message queue. This ensures that QTEVIREF is considered as a request receiving program and that the message is in the job log.

If QTEVIREF is terminated abnormally (it does not itself execute a return), QTEMBKCU is invoked by the machine to clean up the breakpoint data, decrement the breakpoint count, and destroy temporary spaces that may have been created.

7 If there are variables to display, their ODT numbers are retrieved from QTESBVST. QTEMGTVR is invoked to get the variable values. The values returned by QTEMGTVR are passed to QTEMFMT, which presents the breakpoint display.

8 When the display is terminated, QTEVIREF does one of the following:

- If in batch mode and a breakpoint program is specified, invoke the program. After the program returns, return to the interrupted program.

- If in batch mode and a breakpoint program is *not* specified, return to the interrupted program.

If in interactive mode and the display is terminated by the Enter key, return to the interrupted program. Before returning, QTEVIREF does its own cleanup and clears the invocation exit so that QTEMBKCU is not invoked.

If in interactive mode and the display is terminated by the CF3 key, perform the command entry function (receive a message from the requestor and pass the data to the command analyzer for execution).

If in interactive mode and the display is terminated by the CF1 key, the last request entered before this breakpoint occurred is canceled, by invoking QTECCNRQ.

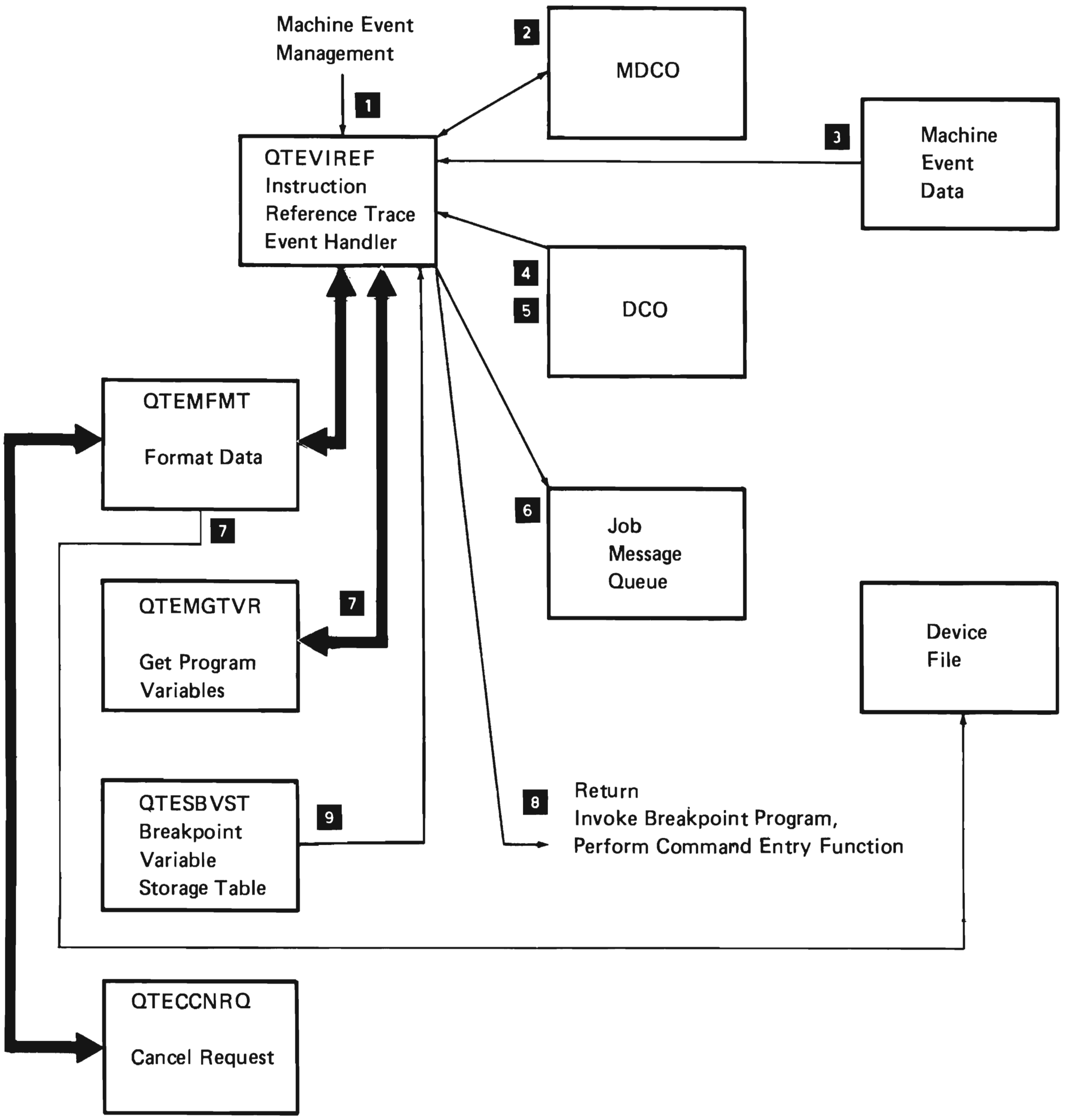

Machine Event Management
1
2
MDCO
QTEVIREF Instruction Reference Trace Event Handler
3
Machine Event Data
4
5
DCO
QTEMFMT Format Data
6
Job Message Queue
7
QTEMGTVR Get Program Variables
7
Device File
QTESBVST Breakpoint Variable Storage Table
9
8
Return Invoke Breakpoint Program, Perform Command Entry Function
QTECCNRQ Cancel Request

Figure TE-20. Instruction Reference Event Handler Overview

Instruction Reference Trace Handling

Figure TE-21 and the following text describe the operation of the instruction reference trace handler.

1. Addressability to the MDCO and DCO is obtained.
2. All QTESTIOs, one for each active trace range, are searched for the instruction number. When the instruction is found, a trace record for the instruction is formatted and put in the trace data space.
3. If there are variables defined for the trace range, as indicated by the presence of a QTESTDO, QTEMGTVR is invoked to get the current values of the variables.
4. If OUTVAR (*CHG) was specified as a parameter on the ADDTRC (add trace) command, the new values are compared to the old values, which were saved in a corresponding QTESTVO.
5. If this is the first tracepoint or if a value has changed, so that the values are new, QTEMFMT is called to format and save the values as additional records into trace data spaces.

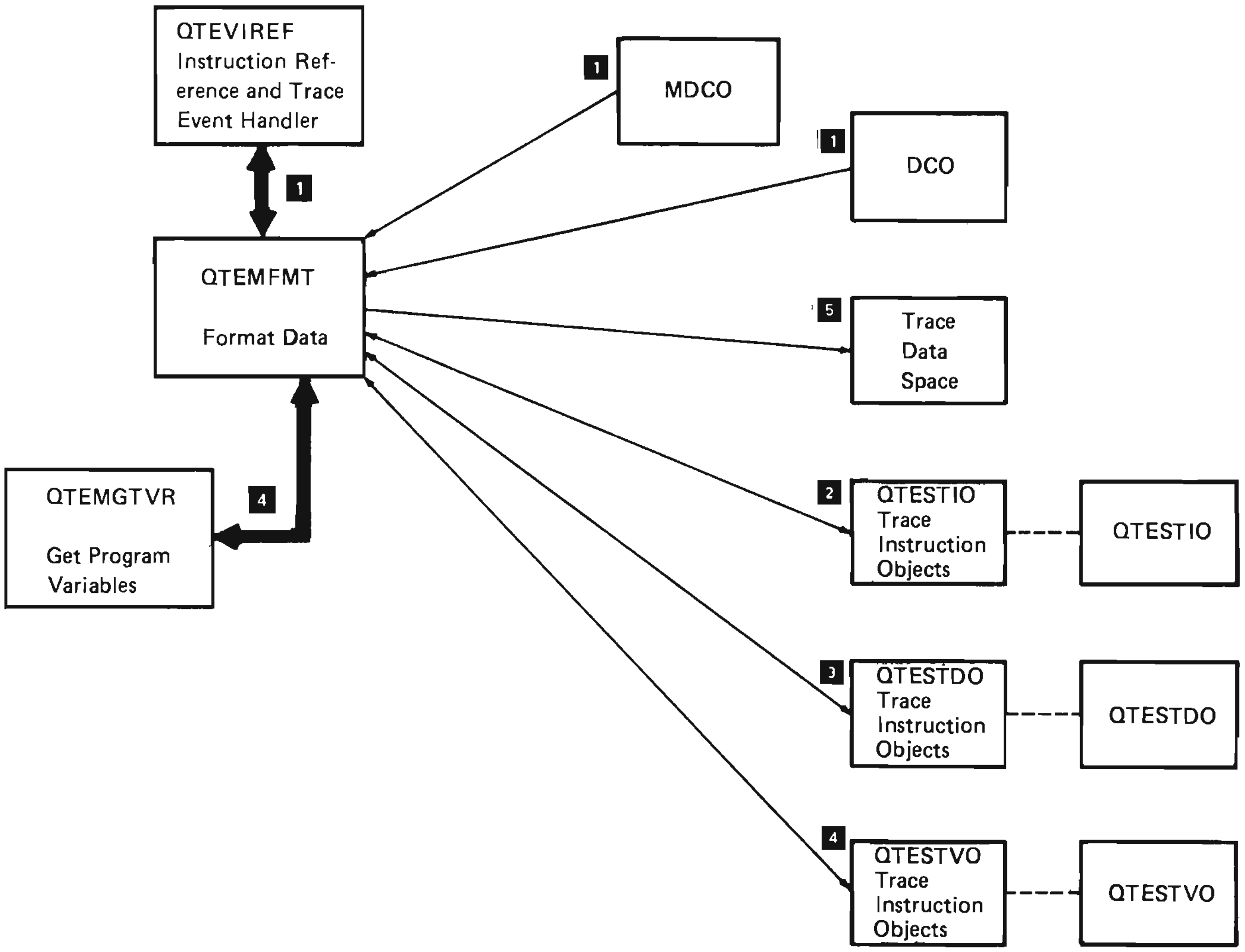

QTEVIREF
Instruction Reference and Trace Event Handler
1
QTEMFMT
Format Data
4
QTEMGTVR
Get Program Variables
1
MDCO
1
DCO
5
Trace Data Space
2
QTESTIO Trace Instruction Objects
QTESTIO
3
QTESTDO Trace Instruction Objects
QTESTDO
4
QTESTVO Trace Instruction Objects
QTESTVO

Figure TE-21. Instruction Reference Trace Handler Overview

Verify Object References

Figure TE-22 and the following text describe the operation of the verify ODV (object directory vectors) reference operation.

1 QTEMVFOR can be called by QTECADBP, QTECADTR, QTECCHVR, QTECCHPT, QTECCHHP, or QTECDSVR. The calling module passes the list of program variable and basing pointer names (or ODV numbers) to be verified to QTEMVFOR as well as a pointer to a space that will contain the return areas for all of the specified variables.

2 Using pointers in the DCO, QTEMVFOR accesses the symbol table, ODV, OES and OMT portions of the program template.

3 QTEMVFOR analyzes each ODV reference, variable name, or basing pointer name from the list that was passed to it. The appropriate return space for each ODV reference is filled in. For variables that pass the verification test, other fields in the return area are filled in. These fields describe the attributes of the variables.

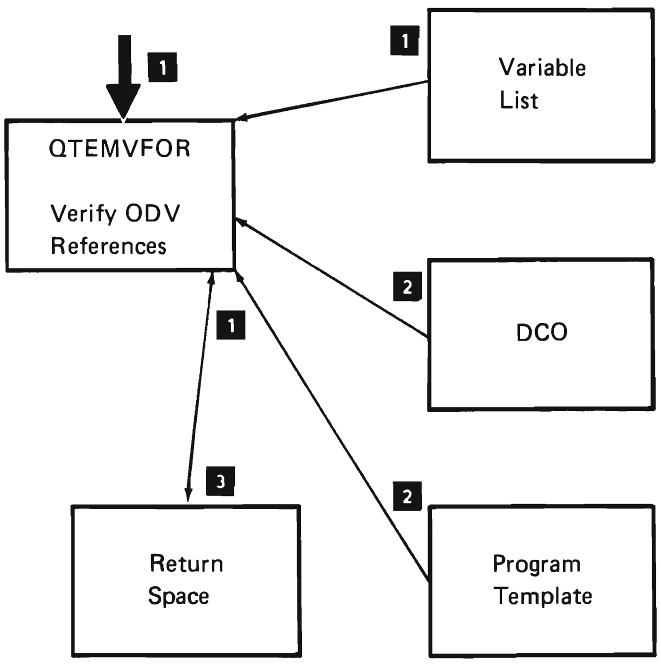


Figure TE-22. Verify Object Directory Vector References Overview

Get Variable Value

Figure TE-23 and the following text describe the get variable value operation.

1 QTEMGTVR can be called by QTECDSVR, or QTEVIREF, to get the current value of variables. QTEMGTVR is passed pointers to the MDCO, the DCO, and descriptors of the specified variables. Also, a system pointer is returned to the space used for returned data.

3 QTEMGTVR calls QTEMLOBJ to obtain addressability to the specified variables.

4 QTEMGTVR creates a space for the return data. Addressability to this space is put in the pointer that was passed to QTEMGTVR (see **1**). The attributes and values are put in the return space, for use by the callers of QTEMGTVR.

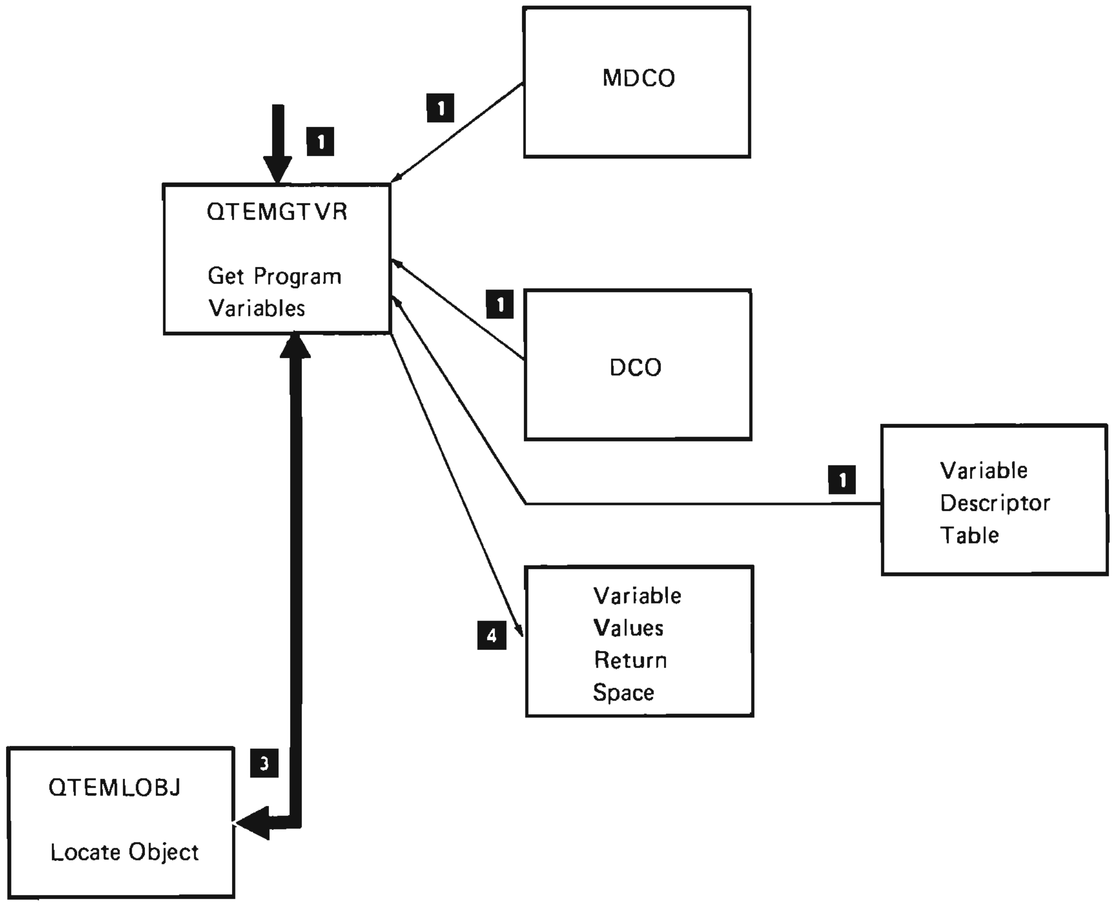


Figure TE-23. Get Variable Value Overview

Locate Object

Figure TE-24 and the following text describe a locate object operation.

1 QTEMGTVR calls QTEMLOBJ to get the addresses of those variables whose values are to be determined, or QTECCHPT, QTECCHHP, or QTECCHVR calls QTEMLOBJ to get the address of the pointer or variable that is to be changed. QTEMLOBJ is passed pointers to the variable descriptor table, the DCO, and PSSA, and PASA entries for the program.

2 Using a pointer from the DCO, QTEMLOBJ locates the OMT (object mapping table) of the program template. QTEMLOBJ locates the address of the specified object using the OMT, the variable, base, and subscript information in the descriptor table. If an error prevents the finding of the address, one or more message identifiers are returned in the descriptor table to report the errors.

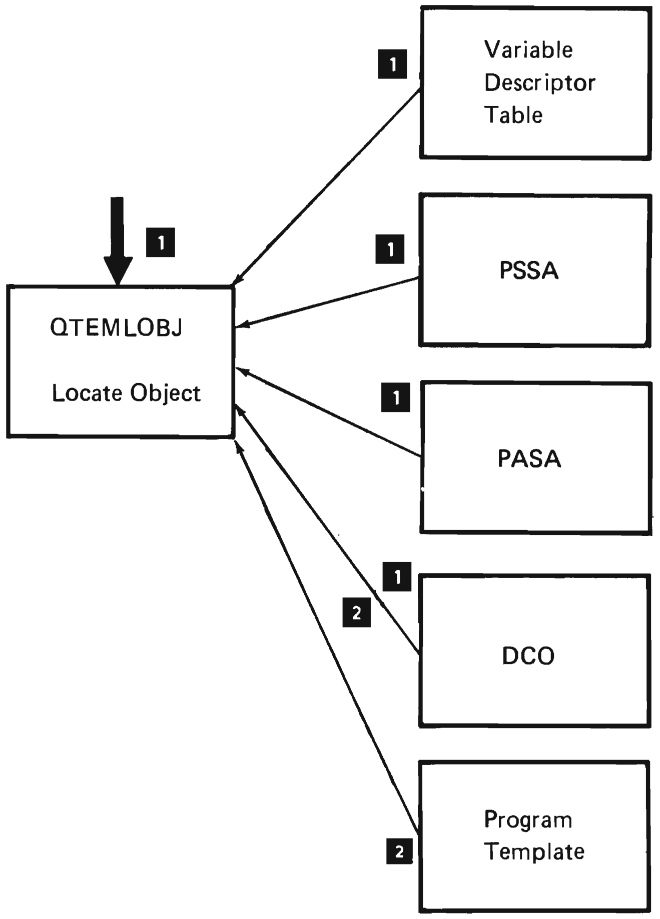


Figure TE-24. Locate Object Overview

Default Exception Breakpoint

Figure TE-25 and the following text describe the overview of the exception breakpoint.

1 When QMHPDEH (of the message handler) determines that there is an unmonitored escape message in an interactive debug job, it calls QTEMDEBP. QTEMDEBP is passed the name of the message queue to which the message was sent and the message reference key.

2 QTEMDEBP, using the information passed to it, receives the message. QTEMDEBP examines the message ID to ensure that it is not one of the internal messages used by the test component. If it is an internal test message, QTEMDEBP returns control to its caller. If the message is *not* an internal message, all of the following steps (3 through 7) are performed.

3 QTEMGTHL retrieves the high-level statement identifier associated with a given machine interface instruction number.

4 A request message is sent to and received from this invocation program message queue. This ensures that QTEMDEBP is considered as a request receiving program and the message is in the job log.

5 The MDCO is modified to indicate that a breakpoint has occurred.

6 QTEMFMT is invoked to display the breakpoint message.

7 If the display is terminated by the Enter key, QTEMDEBP returns control to its caller. Before returning, QTEMDEBP does its own cleanup and cancels the invocation exit so that QTEMBKCU is not invoked. If the display is terminated by the CF3 key, the command entry function is provided. If the display is terminated by the CF1 key, the last request entered is canceled by invoking QTECCNRQ.

Also, QTEMDEBP sets QTEMBKCU as an invocation exit. If QTEMDEBP is terminated abnormally (it does not itself execute a return), QTEMBKCU is invoked to clean up the breakpoint data, decrement the breakpoint count, and destroy temporary spaces that may have been created.

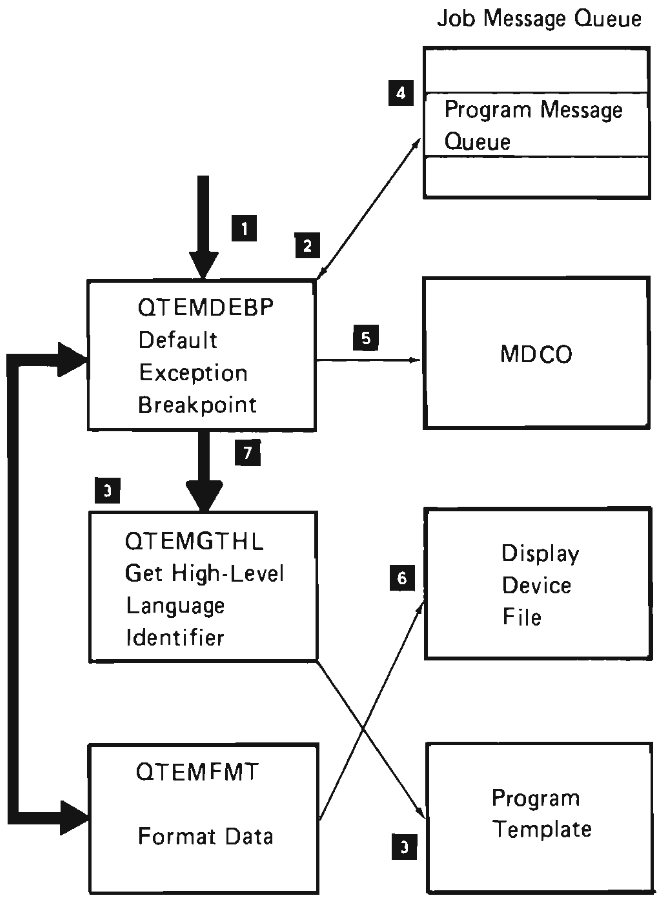


Figure TE-25. Exception Breakpoint Overview

INTRODUCTION

The commitment control component of the CPF (control program facility) provides the user a means of defining and processing a number of changes to data base files as a single unit of work (or transaction). A transaction is defined as a group of changes made to data base files that appear to be a single change.

When data base files are placed under commitment control, all changes associated with a single transaction are completed before any of the changes are written permanently to auxiliary storage.

The following functions make up the commitment control component:

- Command processing programs:
  - Establish commitment control
  - Commit data base changes
  - Rollback data base changes
  - End commitment control
  - Display job (commitment display support)
- IPL (initial program load) recovery processing
- Open processing
- Close processing

GENERAL OVERVIEW

Commitment Control Modules

The commitment control component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QTNCLOSE–Commitment Control Close Processing: This module does close processing related to commitment control.

-->QTNCMT–Commit (COMMIT)[1]: This module commits entries and saves new commitment IDs.

-->QTNCTL–Establish Commitment Control (BGNCMTCTL)[1]: This module establishes a commitment control environment from the parameters on the Begin Commitment Control command.

> QTNNTRS–Resolve to Notify Object and Check Authority: This module resolves to the notify object and verifies that the user is authorized to add data to the notify object.

-->QTNDSPY–Commitment Control Display: This module is invoked by display job support to present commitment control information for the specified job.

-->QTNEND–End Commitment Control (ENDCMTCTL)[1]: This module terminates the commitment control environment from the parameters on the End Commitment Control command, or as part of process termination.

> QTNNTFY–Commitment Control Notify Object Handler: This module places a commitment ID in the user-defined notify object. The notify object may be message queue, data area, or data base file.

> QTNNTRS–Resolve to Notify Object and Check Authority: This module resolves to the notify object and verifies that the user is authorized to add data to the notify object.

-->QTNIPL–Commitment Control Initial Program Load Recovery Handler: This module performs initial program load clean-up for machine interface commitment blocks.

> QTNNTFY–Commitment Control Notify Object Handler: This module places a commitment ID in the user-specified notify object. The notify object may be a message queue, data area, or data base file.

---

[1]This module is a CPP (command processing program).

-->QTNOPEN–Commitment Control Open Processing: This module does open processing related to commitment control.

-->QTNROLLB–Rollback (ROLLBACK)[1]: This module removes changes that have been made to files related to commitment control.

Begin Commitment Control Command

A Begin Commitment Control (BGNCMTCTL) command establishes commitment control. All data base files placed under commitment control must be journaled to the same journal. When changes are made to these data base files, entries are placed in the journal for each change generated by the transaction. When the transaction is completed, the program executes a commit operation that places a separate entry on the journal to identify that the changes generated by the transaction are committed.

Commit Command

A Commit (COMMIT) command causes all of the data base changes that have occurred since the last commit operation to be written permanently to the file member. When a Commit command is executed the following occurs:

- All records that were locked during the processing of the transaction are released.
- All changes to access paths that were updated to reflect data base changes are made permanent.
- The commitment ID if specified is saved for use at recovery time.
- Entries are placed on the journal to indicate that the changes under commitment control have been made.
- I/O feedback area and I/O buffers are unchanged.

Rollback Command

A Rollback (ROLLBACK) command removes changes that have been made to files under commitment control since the last commit operation. When a Rollback command is executed the following occurs:

- All changes made since the last commit boundary are removed from the data base.
- All data base records that were locked are unlocked and made available to other users.
- An entry is placed on the journal to indicate that a rollback operation occurred.
- I/O feedback area and I/O buffers remain unchanged.

End Commitment Control Command

An End Commitment Control (ENDCMTCTL) command ends the use of commitment control. If there are any uncommitted changes when the End Commitment Control command is executed, an escape message is sent.

[1]This module is a CPP (command processing program).

Commitment Display Support Using the Display Job Command

The Display Job (DSPJOB) command provides a screen interface to commitment display support. The display job interface locates the job requested on the Display Job command and provides job information and a pointer to the work control block table entry for that job.

Commitment display support provides information pertaining to a process under commitment control. This information includes:

- Journal information
  - Name and library
  - Commitment ID
- Commitment control environment
  - Notify object name, library, member name, and type
  - Lock level
- Commitment control current status
  - Total number of commit operations performed
  - Total number of rollback operations performed
- Files under commitment control
  - Name, library, member name, and status
  - Current number of committed data base changes
  - Current number of rolled back data base changes
  - Current number of pending data base changes

Establishing Commitment Control Overview

Figure TN-1 and the following text describe an overview of establishing commitment control.

1 The command analyzer decodes a Begin Commitment Control (BGNCMTCTL) command and control is transferred to QTNCTL.

2 QTNCTL calls QTNNTRS and a pointer to the notify object parameter list is passed. QTNNTRS resolves to notify object and verify authority.

3 QTNCTL then creates a commitment definition and copies information from the command to the commitment definition.

4 A pointer is set in the work control block to point to the commitment definition, for use by other commitment control modules.

5 Control is returned to the caller.

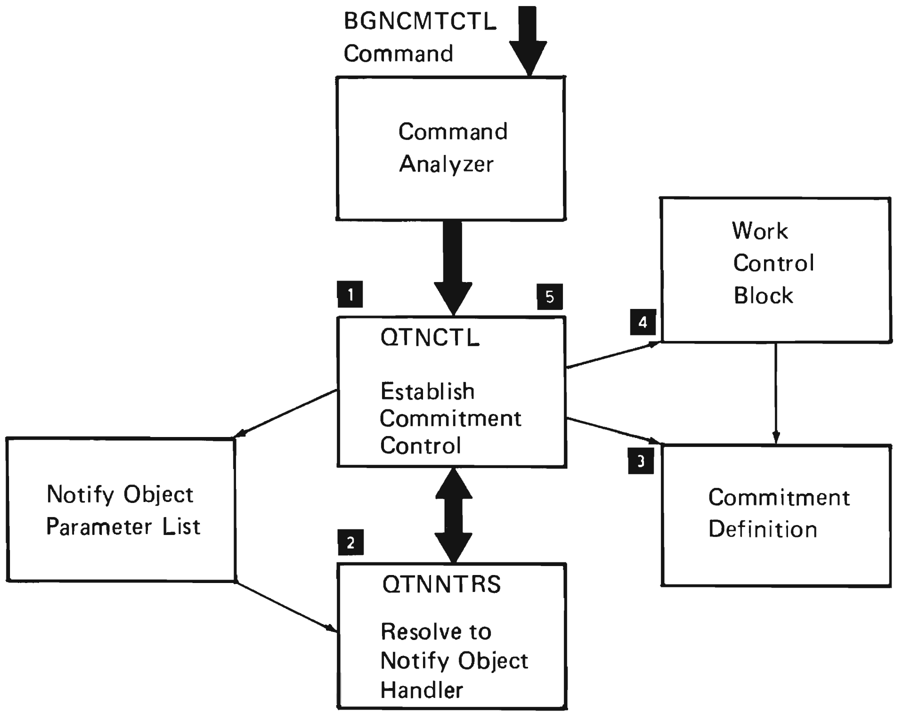


Figure TN-1. Establishing Commitment Control Overview

Open Processing Overview

Figure TN-2 and the following text describe an overview of open processing for commitment control.

1 The user program requests a data base open via the QDMCOPEN module of common data management. When a data base member is opened, the UFCB (user file control block) which points to the active cursor in the ODP (open data path) is passed to QTNOPEN.

2 QTNOPEN processes the user request and saves the file name, member name, and library name. The commitment definition is updated to indicate that the file is open.

3 On the first call to QTNOPEN, the commitment definition is journaled to the same journal as the member passed in the input, and attached to the machine interface process.

4 Control is returned to the user program.

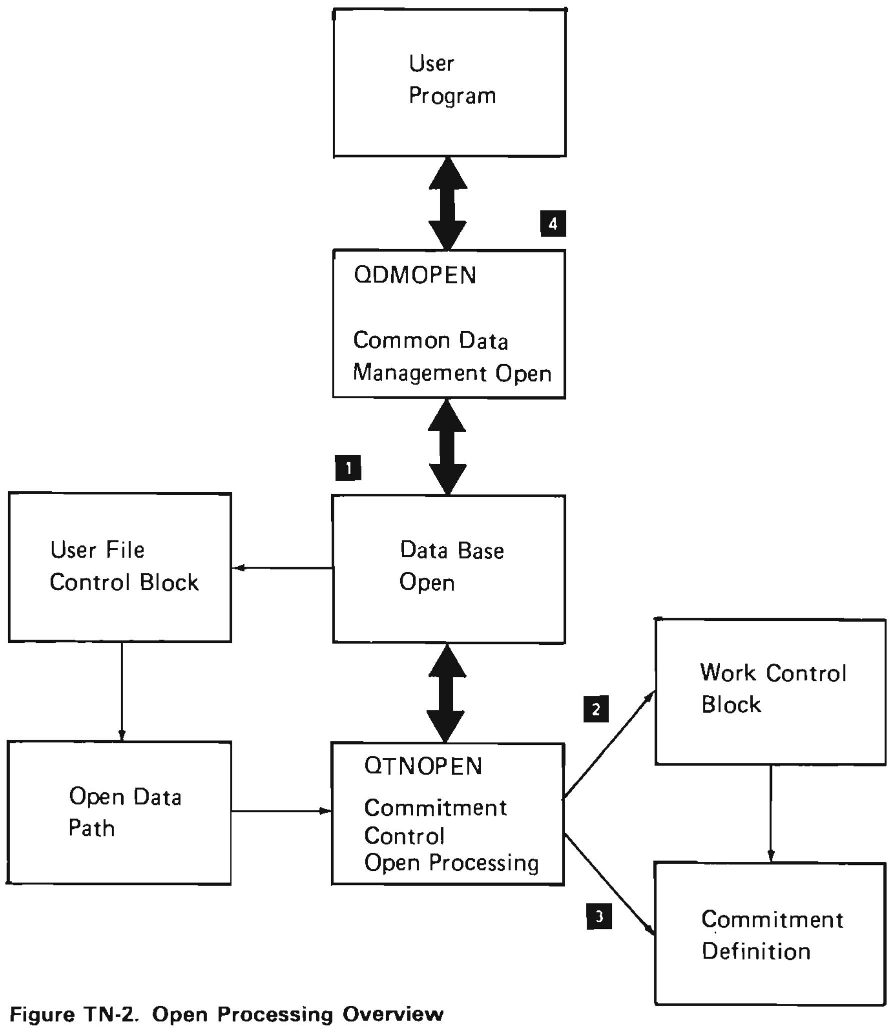


Figure TN-2. Open Processing Overview

Commit Overview

Figure TN-3 and the following text describe an overview of a commit transaction.

1 The command analyzer decodes a Commit (COMMIT) command and control is transferred to QTNCMT, or the module is called by the ?COMMIT macro.

2 QTNCMT provides the CPF interface to the machine interface commit instruction, updates the number of committed changes for each file, and increments the total commitment count.

3 QTNCMT will also save positioning information for each ODP, in case a rollback is performed by the user.

4 Control is returned to the caller.

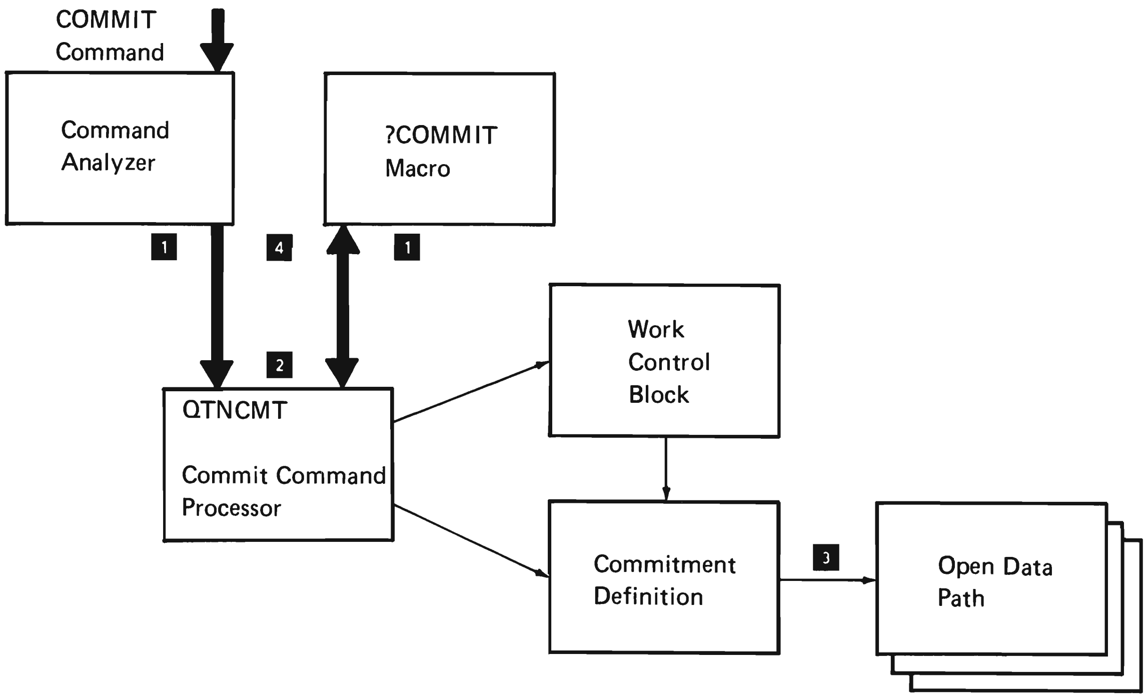


Figure TN-3. Commit Overview

Rollback Overview

Figure TN-4 and the following text describe an overview of a rollback operation.

1 The command analyzer decodes a Rollback (ROLLBACK) command and control is transferred to QTNROLLB, or the module is called by the ?ROLLBACK macro.

2 QTNROLLB provides the CPF interface to the machine interface rollback instruction. It also updates the number of rollbacks for each file, and removes closed files from the file list.

3 Information saved by QTNCMT is used to reposition each ODP.

4 Control is returned to the caller.

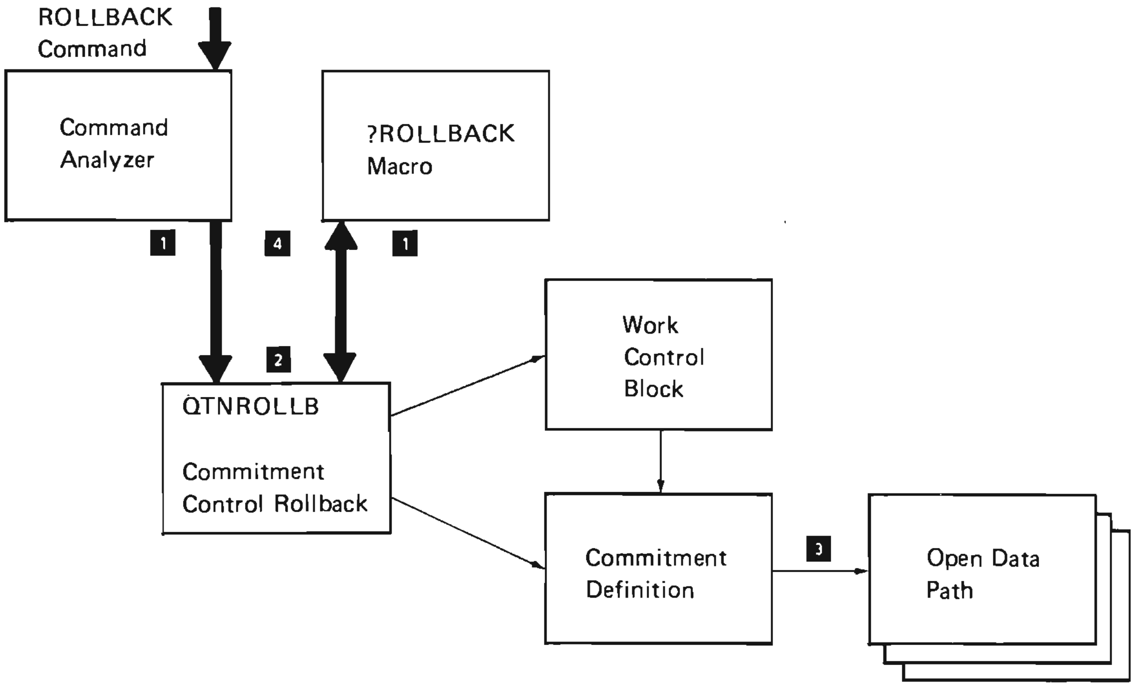


Figure TN-4. Rollback Overview

Display Support Overview

Figure TN-5 and the following text describe an overview of display support.

1 QTNDSPY is invoked by the ?TNDSP macro.

2 The requested information is retrieved by the display job structure from the commitment definition.

3 QTNDSYP formats the display screen and/or prepares to print the commitment information. QTNDSYP uses the display job print file to print the information.

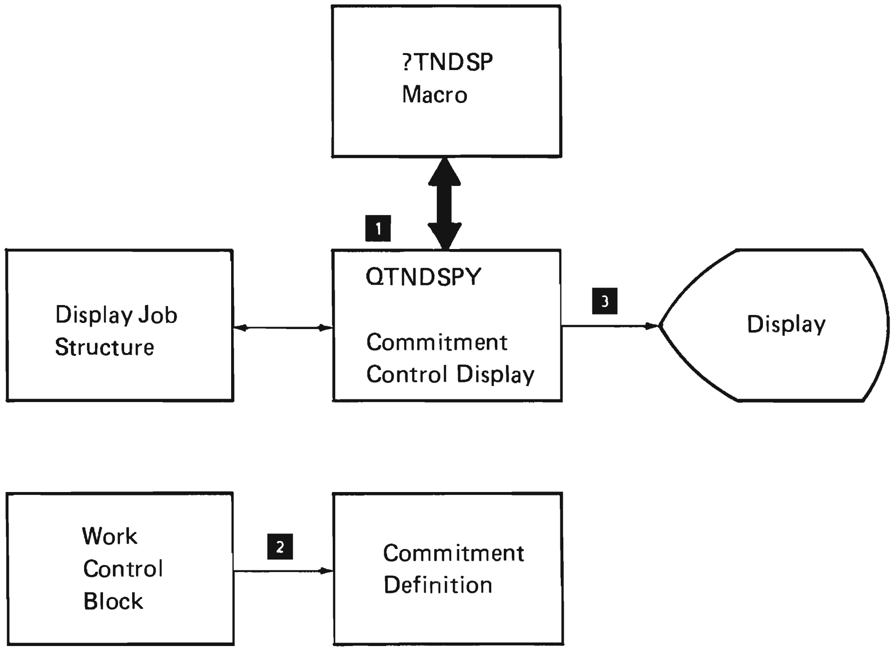


Figure TN-5. Display Support Overview

End Commitment Control Overview

Figure TN-6 and the following text describe an overview of ending a transaction.

1. The command analyzer decodes the End Commitment Control (ENDCMTCTL) command and control is transferred to QTNEND, or the module is called by process termination.

2. QTNEND removes all ODPs from the commitment definition, detaches the commitment definition from the process, and destroys it.

3. QTNEND calls QTNNTFY to handle the processing of the commitment control notify object.

4. Control is returned to the caller, or toprocess termination.

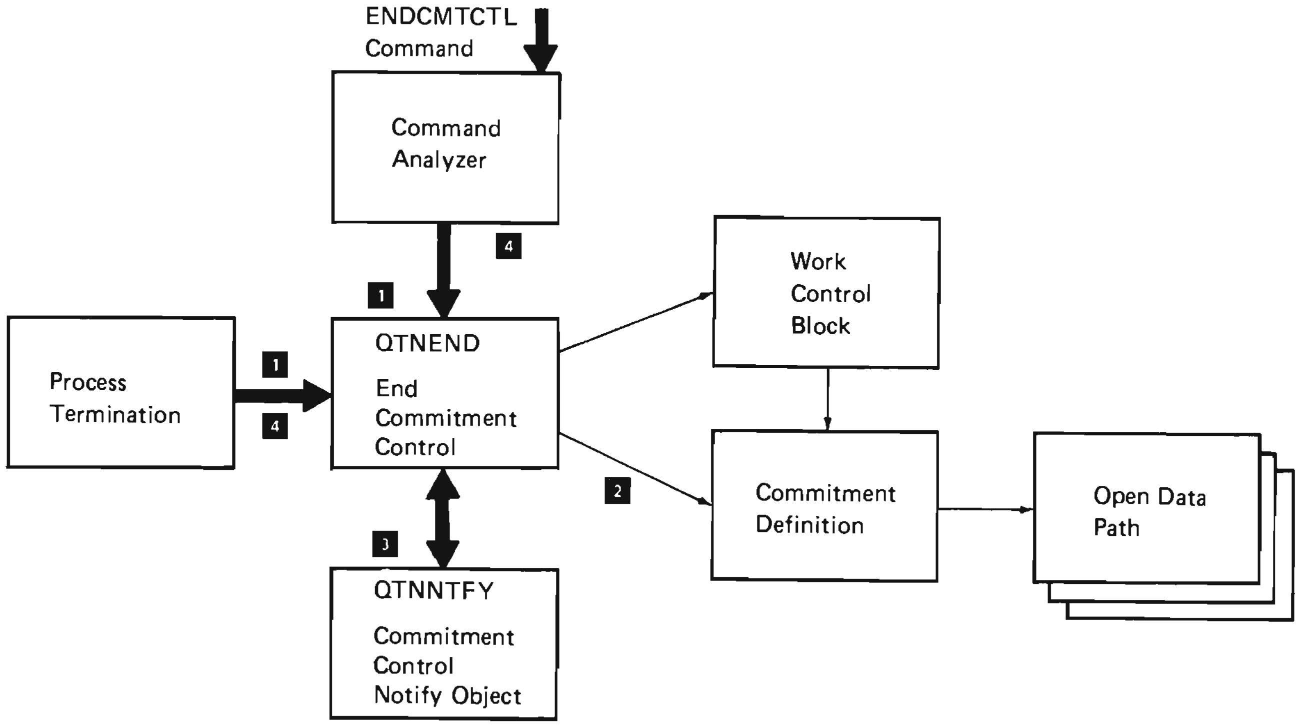


Figure TN-6. End Commitment Control Overview

Notify Object Overview

Figure TN-7 and the following text describe an overview of notify object processing.

1 QTNEND or QTNIPL calls QTNNTFY.

2 QTNNTFY calls QTNNTRS and a pointer to the notify object parameter list is passed.

3 QTNNTFY updates the user-defined notify object with the last user-defined commitment ID. The notify object may be a message queue, data area, or data base file.

4 Control is returned to the caller.

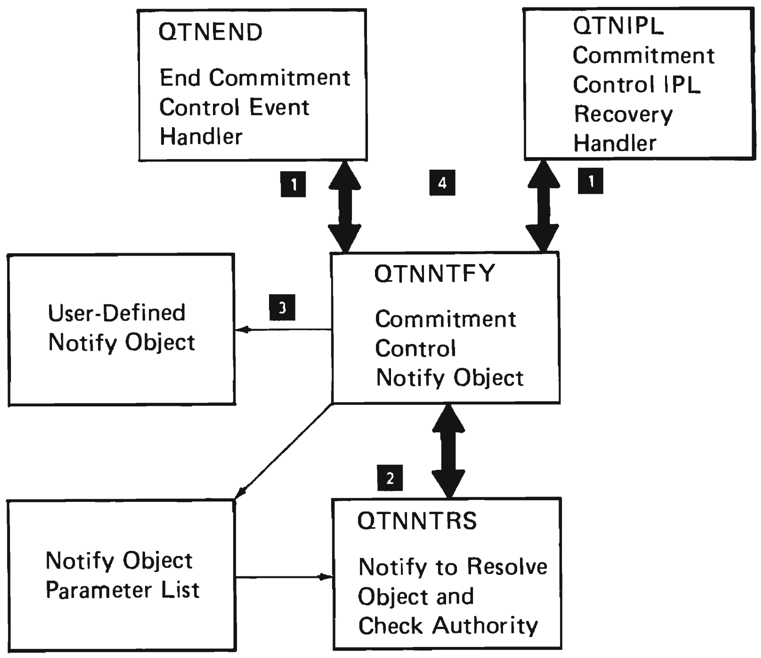

QTNEND
End Commitment Control Event Handler
QTNIPL
Commitment Control IPL Recovery Handler
1
4
1
User-Defined Notify Object
3
QTNNTFY
Commitment Control Notify Object
2
Notify Object Parameter List
QTNNTRS
Notify to Resolve Object and Check Authority

Figure TN-7. Notify Object Overview

Initial Program Load Recovery Overview

Figure TN-8 and the following text describe an overview of initial program load recovery.

1 QRCIMPLN calls QTNIPL. A parameter list is passed that contains a pointer to the object recovery list.

2 Commitment definitions are accessed via the object recovery list and destroyed.

3 QTNIPL calls QTNNTFY for each commitment definition on the object recovery list. QTNNTFY handles the processing of the notify objects.

4 QRECOVERY is searched for any remaining commitment definitions. If any are found they are also destroyed.

5 Control is returned to the caller.

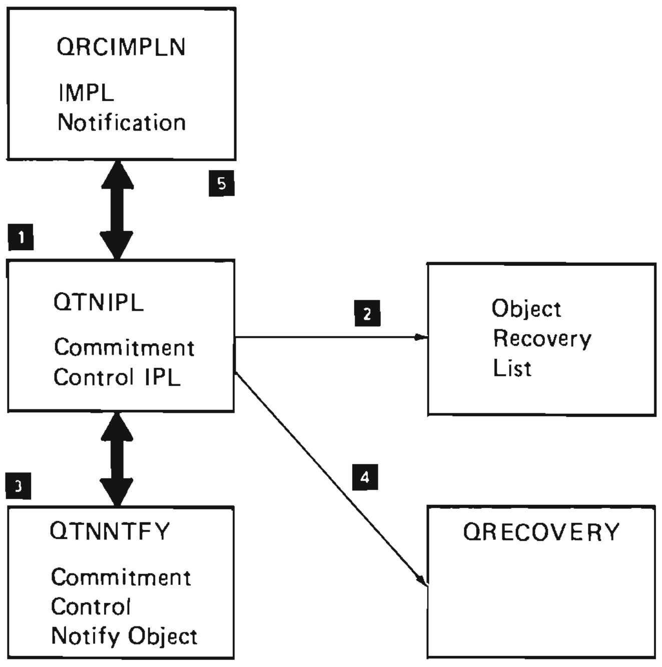


Figure TN-8. Initial Program Load Recovery Overview

Close Processing Overview

Figure TN-9 and the following text describe an overview of close processing for commitment control.

1 The user program requests a data base close via the QDMCLOSE module of common data management.

2 QTNCLOSE processes the user request, removes all ODPs from the commitment definition, and updates the commitment definition to indicate the file is closed.

3 Control is returned to the user program.

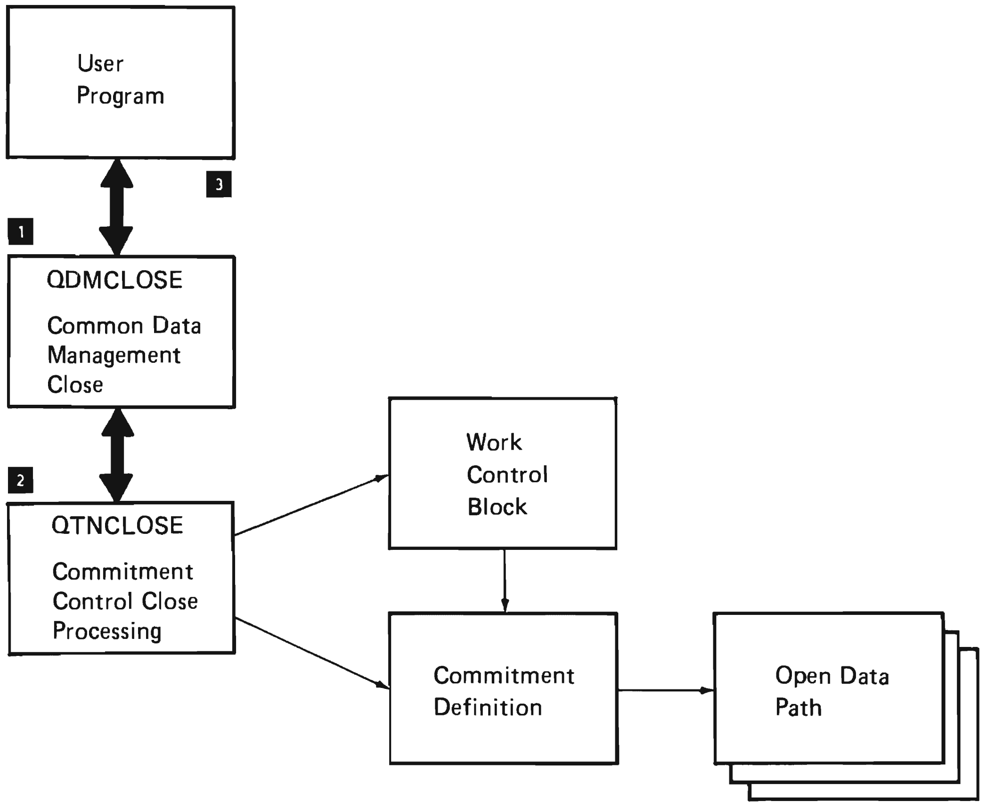


Figure TN-9. Close Processing Overview

INTRODUCTION

The SNA-T3 component of the CPF (control program facility) provides SNA (system network architecture) support for devices that operate as SNA logical unit types 4 (IBM 5256) or 7 (IBM 5251). The 5251 Display Terminal and the 5256 Printer are the devices attached to the System/38 that operate as SNA logical unit types 4 or 7. Therefore, the 5251 function manager and the 5256 function manager components interface with SNA-T3 to perform all of their I/O operations.

GENERAL OVERVIEW

The device support functions manager, either 5251 or 5256, creates request blocks that are used to pass information to and receive information from SNA-T3. The information in the request block that is passed to SNA-T3 from one of the function managers includes a function request code, option bit settings, transmission data length, and the actual transmission data. Information returned to the function manager by SNA-T3 includes the received data length, received data type code, error information, and the received data, if any.

SNA-T3 Modules

The SNA-T3 component consists of the following module:

-->QT3REQIO-SNA-T3 Request I/O: This module provides the SNA control interface for the communication of data, commands, and responses between the caller of this module and the SNA device.

FUNCTIONAL OVERVIEWS

Wait Operation Overview

Figure T3-1 and the following text describe a wait operation.

1 The user program calls the function manager to perform a function.

A If the function is an open or close operation, common data management is called by the function manager user prior to calling the function manager.

2 The function manager performs its requested function and calls QT3REQIO to issue all of the REQIO instructions.

3 In addition to issuing REQIO instructions, QT3REQIO finishes setting up the source/sink request and also enforces SNA protocols. The REQIO instruction is issued to the device I/O manager and QT3REQIO also issues a dequeue with wait to the machine interface response queue.

4 The REQIO is completed and a completion message (feedback record) is placed on the machine interface response queue by the device I/O manager (a machine function).

5 The dequeue instruction is now satisfied. QT3REQIO processes any input data and status received from the feedback record. Control is returned to the function manager. The requested function is finished and control is returned to the function manager user.

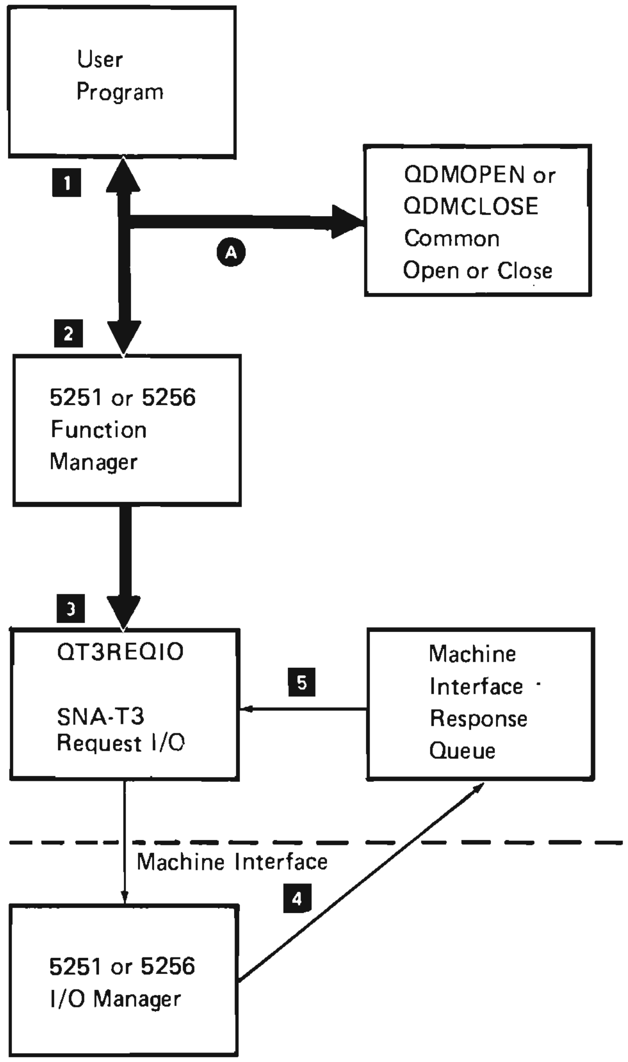

User Program
1
A
QDMOPEN or QDMCLOSE Common Open or Close
2
5251 or 5256 Function Manager
3
QT3REQIO SNA-T3 Request I/O
5
Machine Interface Response Queue
Machine Interface
4
5251 or 5256 I/O Manager

Figure T3-1. Wait Operation Overview

Nowait Operation Overview

Figure T3-2 and the following text describe a nowait operation.

1 The user program calls the function manager to perform a function.

A If the function is an open or close operation, common data management is called by the function manager user prior to calling the function manager.

2 The function manager performs its requested function and calls QT3REQIO to perform all of the REQIO instructions.

3 In addition to issuing REQIO instructions, QT3REQIO finishes setting up the source/sink request and also enforces SNA protocols. The REQIO instruction is issued to the device I/O manager, and control is returned to the function manager.

4 When the I/O manager is finished with the REQIO instruction, it sends a feedback record to the machine interface response queue.

5 The I/O manager also signals an REQIO complete event to the function manager REQIO complete event handler module.

6 The function manager REQIO complete event handler module is invoked. It dequeues the feedback record from the machine interface response queue.

7 QT3REQIO is then called to process any incoming data and to continue enforcing SNA protocols. After all requested functions have been performed, control is returned to the function manager REQIO complete event handler module.

8 The function manager REQIO complete event handler invokes the proper function manager modules to process the data. Control is returned to the event handler.

9 The event handler signals a put-wait or a get nowait complete event to the user.

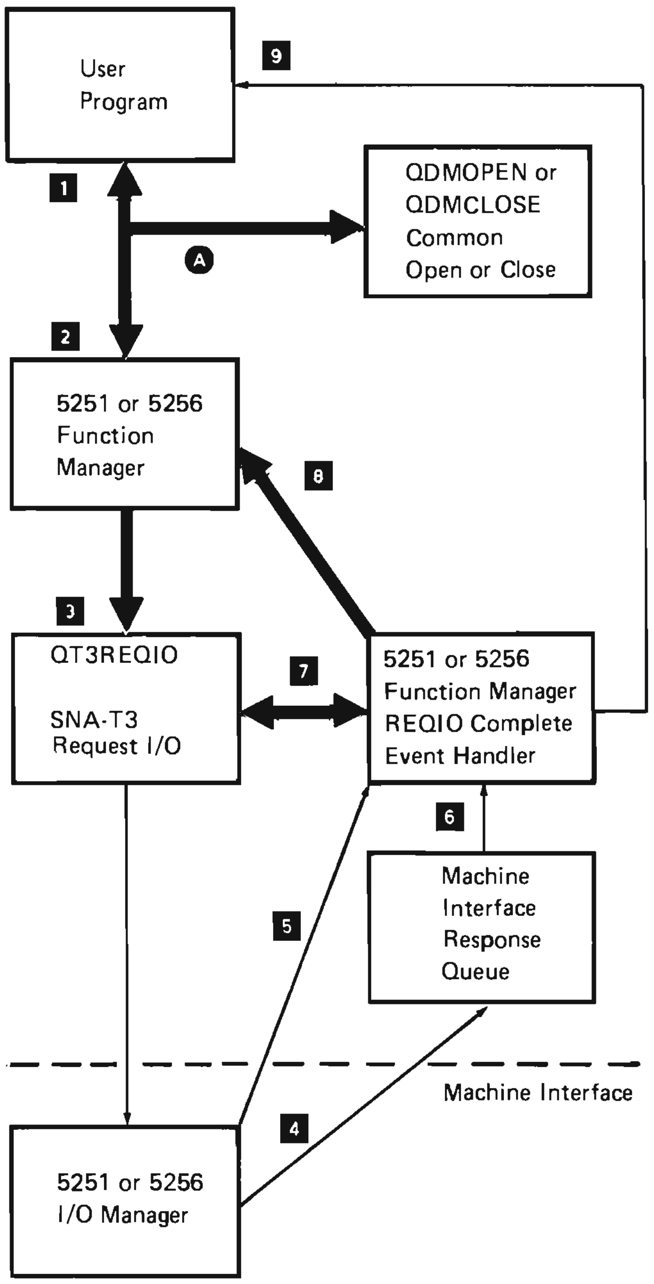

User Program
9
1
QDMOPEN or QDMCLOSE Common Open or Close
A
2
5251 or 5256 Function Manager
8
3
QT3REQIO SNA-T3 Request I/O
7
5251 or 5256 Function Manager REQIO Complete Event Handler
6
Machine Interface Response Queue
5
Machine Interface
4
5251 or 5256 I/O Manager

Figure T3-2. Nowait Operation Overview

Unsolicited Data Operation Overview

Figure T3-3 and the following text describe the handling of unsolicited data.

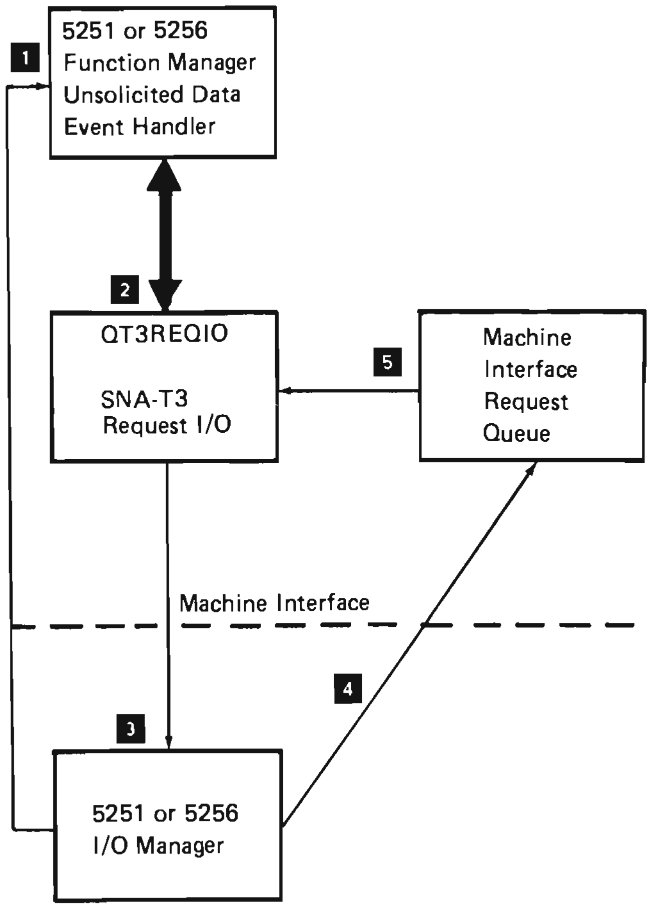
1 An unsolicited data event is signaled by the I/O manager when unexpected data is received from the I/O device. This event causes the function manager unsolicited data event handler module to be invoked.

2 The unsolicited data event handler module calls QT3REQIO to get the unsolicited data.

3 QT3REQIO builds an REQIO instruction and issues it. A dequeue with wait is also issued to the machine interface response queue.

4 The REQIO instruction is completed and a feedback record is put on the machine interface response queue.

5 The dequeue is satisfied and QT3REQIO processes the incoming data as well as enforcing the SNA protocols. Control is returned to the unsolicited data event handler module.



1
5251 or 5256 Function Manager Unsolicited Data Event Handler
2
QT3REQIO
SNA-T3 Request I/O
5
Machine Interface Request Queue
Machine Interface
4
3
5251 or 5256 I/O Manager

Figure T3-3. Unsolicited Data Operation Overview

INTRODUCTION

The work control component of the CPF (control program facility), unlike other CPF components that provide a common function, is instead a collection of miscellaneous and separate functions. The functions that are provided by work control are:

- Start CPF
- System arbiter process
- Logical unit services process
- Commands
  - Start a subsystem
  - Terminate a subsystem or system
  - Allocate or deallocate an object
  - Display information about jobs, subsystems, system, system status, and object locks
  - Class
  - Data areas
  - System values
  - Sign-off
  - Network attributes
- System date and time support
- System timer support
- Storage pool and multiprogramming-level resource management
- Work control block table and work control block maintenance
- Machine status and resource event handling
- Reclaim resources function

GENERAL OVERVIEW

Work Control Modules

The work control component consists of the following modules:

Note: Modules identified by an arrow (-->) are entry modules into the component. Indentation of a module shows its dependency on a preceding module.

Start CPF Function Modules

-->QWCIINSR–Initial CPF Process: This module creates a standard CPF job structure and then initiates the start CPF process into that job structure.

> QWCIPDEH–Initial CPF Process Default Exception Handler: This module provides the initial CPF process with the default exception handling for all unplanned exceptions that might occur.
>
> QWCSCRLR–Create/Convert System Value Object Routine: This module converts the system value object to a new release/modification level following installation of CPF. It adds new system values without affecting any changes made by the user to existing system values. QWCSCRLR also re-creates the system value object if it becomes damaged or destroyed.
>
> QWCISCFR–Start CPF Process: This module is the start CPF process problem phase program.
>
> > QWCICLSR–WCBT Cleanup Routine: This module ensures that the WCBT (work control block table) and related objects that constitute the permanent job structure are in a condition to support normal operation.

System Arbiter Process Modules

-->QWCAINAR–System Arbiter Initiation Phase Program: This module is the initiation phase program of the system arbiter process (QSYSARB). It creates the monitors for events handled in the system arbiter process.

QWCAMNAR–System Arbiter Problem Phase Program: This module is the mainline program of the system arbiter process and basically waits to be interrupted by an event.

-->QWCAHNAE–System Arbiter Input Event Handler: This module is the event handler for the system arbiter input event. It determines what function is to be performed and calls the appropriate module.

Logical Unit Services Process Modules

-->QWCLINSR–Logical Unit Services Initiation Phase Program: This module creates the monitors for events handled in the logical unit services process (QLUS).

QWCLMNSR–Logical Unit Services Problem Phase Program: This module waits to be interrupted by an event.

Start Subsystem Modules

-->QWCCSUUC–Start Subsystem (STRSBS)[1]: This module is the command interface for starting a subsystem.

QWCASUUM–Start Subsystem Message Processor: This module starts a subsystem monitor process and executes in the system arbiter process.

Subsystem and System Termination Modules

-->QWCCSDUC–Terminate Subsystem (TRMSBS)[1]: This module performs the portion of the terminate subsystem function that is executed in the requester process.

-->QWCCSDSC–Terminate System (TRMCPF and PWRDWNSYS)[1]: This module handles the portion of the terminate system function that is executed in the requester process.

QWCASDUM–Terminate Subsystem Message Processor: This module is a message processor that performs the serial portion of the Terminate Subsystem command and executes in the system arbiter process.

QWCAT1TE–Terminate Subsystem Timer Event Handler: This module handles the timer event that indicates the expiration of the specified delay time associated with the Terminate Subsystem command. This module executes in the system arbiter process.

QWCASCUE–Monitor Process Termination Event Handler: This module handles the process terminate event for a subsystem monitor process or the logical unit services process, and the CPF-defined event that indicates the controlling subsystem has reached the restricted state. This module executes in the system arbiter process.

QWCASDSM–Terminate System Message Processor: This module performs the serial portion of a terminate system type command. This module executes in the system arbiter process.

QWCAT2TE–Terminate System Timer Event Handler: This module processes the timer event that indicates the expiration of the specified delay time associated with a terminate system type command. This module executes in the system arbiter process.

QWCITRSE–System Arbiter Process Termination Event Handler: This module handles the process termination event for the system arbiter process. This module executes in the start CPF process.

Allocate Object and Deallocate Object Modules

-->QWCCALOC–Allocate Object (ALCOBJ)[1]: This module processes the Allocate Object command.

-->QWCCDAOC–Deallocate Object (DLCOBJ)[1]: This module processes the Deallocate Object command.

[1]This module is a CPP (command processing program).

The following module is used by the Allocate Object and Deallocate Object modules:

QWCSADVR–Allocate/Deallocate Input Validity Check Routine: This module validity checks the command input to the allocate and deallocate object commands.

Display Status Information Modules

-->QWCCDSJC–Display Job (DSPJOB)[1]: This module processes the Display Job command.

QWCCDSIC–Display Program Stack Module: This module is called by the Display Job command; it handles the program stack option of the Display Job command.

QWCSDSIM–Display Program Stack Message Processor: This module obtains information necessary to display the program stack and executes in the process of the job being displayed.

QWCSDSKS–Display Job Locks Setup: This module is called by QWCCDSJC to perform setup for the Display Job command.

QWCSDSKC–Display Job Locks Routine: This module displays the locks held by the job for the Display Job command.

-->QWCCDSSC–Display System (DSPSYS)[1]: This module processes the Display System command.

-->QWCCDSTC–Display System Status (DSPSYSSTS)[1]: This module processes the Display System Status command.

-->QWCCDSUC–Display Subsystem/Submitted Jobs (DSPSBS/DSPSBMJOB)[1]: This module processes the Display Subsystem/Submitted Jobs command.

QWCSHNCR–Handle Command: This module is called by the Display Subsystem/Submitted Jobs CPP module, and the Display System CPP module; it handles options selected on the CPPs interactive displays.

QWCSDSFR–Display Spooled Files Interface: This module is called by QWCSHNCR to display spooled files.

-->QWCCDSKC–Display Object Locks (DSPOBJLCK)[1]: This module processes the Display Object Locks command.

QWCSDSKR–Additional Lock Display Routine: This module displays shared member locks for data base files.

-->QWCCDSAC–Display Active Jobs (DSPACTJOB)[1]: This module processes the Display Active Jobs command.

QWCSHACR–Handle Command: This module is called by the Display Active Jobs CPP, the Display Object Locks CPP, and QWCSDSKC to handle options selected on the interactive displays.

QWCSDSAR–Display Active Jobs Exclude Routine: This module is called by QWCSHACR to exclude a job from the active jobs display. This module is also called by the Display Active Jobs CPP to rebuild the subfile without excluded jobs.

QWCSHCFR–Handle Commands from CF Keys: This module is called by Display Active Jobs CPP, Display System CPP, and Display System Status CPP to handle commands invoked by CF keys.

Class Support Modules

-->QWCCCRCC–Create Class (CRTCLS)[1]: This module creates and initializes a class object.

-->QWCCDSCC–Display Class (DSPCLS)[1]: This module processes the Display Class command.

[1]This module is a CPP (command processing program).

System Value Support Modules

-->QWCCCHLC–Change System Value (CHGSYSVAL)[1]: This module processes the Change System Value command.

QWCAUPRM–Update System Values Message Processor: This module is invoked whenever the user changes the system part of the library list, user part of the library list, time of day (to collect statistics on communications lines), total number of additional jobs, number of additional active jobs, job spool area, additional spool area, date, year, month, day, time, hour, minute, or second system value. QWCAUPRM executes in the system arbiter process.

-->QWCCDSLC–Display System Value (DSPSYSVAL)[1]: This module processes the Display System Value command.

-->QWCSRTLR–Retrieve System Value Routine (RTVSYSVAL)[1]: This module is used to retrieve information relating to a particular system value. It also processes the Retrieve System Value command.

Network Attributes Support Modules

-->QWCCCHNC–Change Network Attributes (CHGNETA)[1]: This module processes the Change Network Attributes command.

-->QWCCDSNC–Display Network Attributes (DSPNETA)[1]: This module processes the Display Network Attributes command.

Data Area Support Modules

-->QWCCCRVC–Create Data Area (CRTDTAARA)[1]: This module processes the Create Data Area command.

-->QWCCCHVC–Change Data Area (CHGDTAARA)[1]: This module processes the Change Data Area command.

-->QWCCDSVC–Display Data Area (DSPDTAARA)[1]: This module processes the Display Data Area command.

-->QWCSRTVR–Retrieve Data Area Pointer Routine: This module retrieves the address of the particular data area.

Sign-Off Support Module

-->QWCCLFEC–Sign-Off (SIGNOFF)[1]: This module processes the Sign-Off command.

System Timer Support Modules

-->QWCAROTE–System Timer Rollover Event Handler: This module maintains the occurrence of a timer event at a 24 hour interval, records the time of day clock value for the start of the next day, and updates the date system value.

System Date and Time Support Modules

-->QWCSVRDR–Verify Date System Support Routine: This module verifies that an input date is valid.

-->QWCSVRTR–Verify Time System Support Routine: This module verifies the elements of an input time value.

-->QWCSCDFR–Change Date Format Support Routine: This module converts a date from one specified format to another specified format and optionally inserts separators in the date.

-->QWCSCVDR–Convert System Date Routine: This module converts a date-time value in some standard format to a time of day value in machine clock format.

-->QWCSCVTR–Convert System Time Routine: This module converts a machine clock binary value to a character date-time format.

[1]This module is a CPP (command processing program).

System Resource Support Module

-->QWCARQRM–Request Resource Message Processor: This module processes the system arbiter input event with the request resources subtype. QWCARQRM allocates system resources (storage pools, storage, and MPL) to requesting subsystems.

WCBT Maintenance Support Module

-->QWCAHNTM–Handle WCBT Message Processor: This module handles the condition of no available WCBT entries or job numbers available for assignment to incoming jobs. This module executes in the system arbiter process.

Machine Event Handling Modules

-->QWCAMCKE–Machine Event Handler: This module handles machine-signaled events, such as the machine check event, process events of maximum CPU time, maximum storage exceeded, and power switched to/from auxiliary events.

-->QWCARSRE–Reserved Storage Released Event Handler: This module handles the reserved storage released event. QWCARSRE executes in the system arbiter process.

-->QWCATARE–Machine Resources and Resources Timer Event Handler: This module handles machine-signaled events, such as the auxiliary storage threshold reached event, machine ineligible state event, and MPL class threshold event. QWCATARE also handles the repeating timer event for the auxiliary storage threshold notification.

Reclaim Resource Module

-->QWCCRCRC–Reclaim Resources (RCLRSC)[1]: This module processes the Reclaim Resources command.

START CPF

The CPF can be started in one of two ways: either an AIPL (alternative initial program load) or an IMPL (initial microprogram load). With the rotary switches set at either of these two types, pressing the Load Key (or Power key, if not powered on) causes the initialization of the machine and the starting of the CPF.

AIPL

The AIPL method of starting the CPF causes the machine to initialize itself and then start the CPF installation function. The CPF installation function executes in a machine interface process and installs the CPF on the system. The installation function then establishes the PDT (process definition template) needed for the initial CPF process. A copy of this PDT is stored as a machine attribute, to be used during IMPL. The CPF installation function then initiates the initial CPF process and terminates itself.

IMPL

The IMPL method of starting the CPF causes the machine to initialize itself and then initiate the initial CPF process using the PDT that was installed during an earlier AIPL. IMPL is the common method of starting the CPF.

INITIAL CPF PROCESS

The initial CPF process is a transition from the machine execution environment to that of the CPF. Normal operation of the CPF requires the CPF process data structures, such as the WCB (work control block), JMQ (job message queue) and, DMCQ (data management communication queue). The initial CPF process establishes a standard CPF job structure and the necessary environment for a standard CPF process. The start CPF process is then initiated using the previously established environment and job structure.

The initial CPF process attempts to establish contact with the basic CPF structures that are required to allow this part of the CPF to function. Among these necessary CPF structures are libraries, programs, LUDs, (logical unit descriptions), and data structures. The necessary CPF data structures fall into two categories: those data structures that cannot be re-created if they do not exist and those data structures that can be re-created if they do not exist. The re-creation of a data structure might mean the loss of some operational data, but the system can be assisted in the recovery of such data by the user.

If, during the execution of the initial CPF process, a required CPF structure cannot be located and it cannot be recreated (it must be reinstalled in an AIPL sequence), the initial CPF process will terminate machine processing. If an error condition occurs that cannot be handled by the initial CPF process, machine processing is terminated.

The normal function of the initial CPF process initiates the start CPF process using the standard CPF job structure and then terminates itself. Figure WC-1 shows the relationships of the elements of starting the CPF.

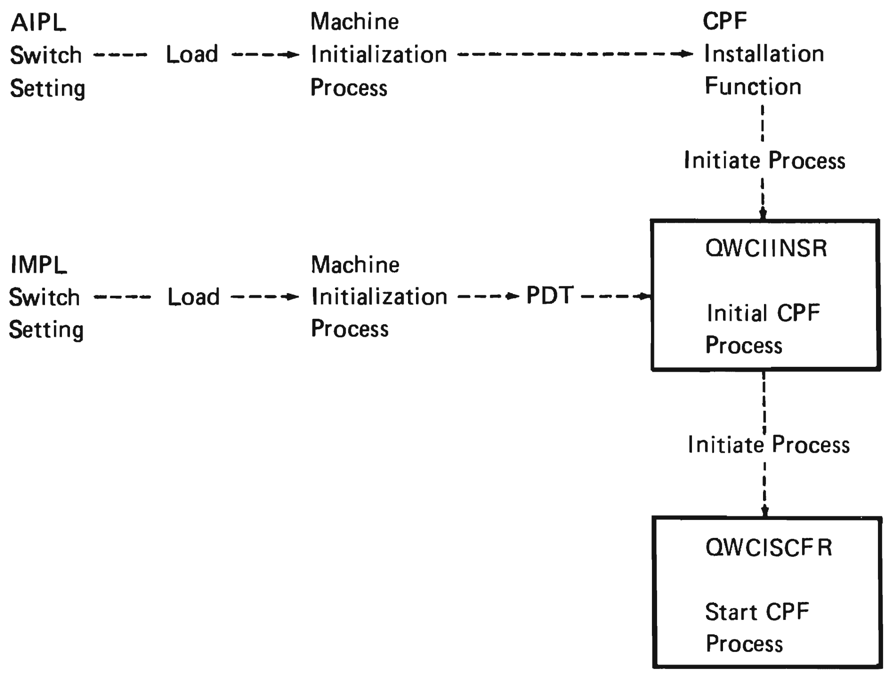


Figure WC-1. Functional Relationships of Starting the CPF

START CPF PROCESS

The start CPF process controls the execution and directs the functions that actually bring up the CPF. These functions, listed in the order that they are executed, are:

1. Call the system logging function to initialize the system log queues and to establish the necessary event monitors to provide the system log support during the start CPF function.

2. Determine if the system was started by an operator that pressed the Load or Power key, or if the system was automatically restarted. If the system was started by an operator pressing a key, determine if the system console is available for a user session and vary it on if it is operational.

3. Determine if the start CPF portion of the CPF installation is required, and call the routine to perform the final CPF installation functions. This function requires an interface with the user through the system console. This function is necessary only after the user has performed the CPF installation function.

4. Present the sign-on display to the user on the system console (assuming a user-initiated system start) and verify that the user sign-on is authorized to perform the start CPF process.

5. After a user has signed on, determine if there are any system program changes to be installed (this also includes the removal of program changes that were installed earlier and remain in the system) and call the service component to perform this function. To perform this function will require that the service component interface with the user through the system console.

6. Display the start CPF prompt so that the user can set the initial system date, time, and the control indicators that are used during the rest of the start CPF process.

7. Call the configuration menu routine if it was requested by the user on the start CPF prompt. This menu is used to establish or alter the system I/O configuration, or to change some system values before the system is activated.

8. Call the data base recovery routine to process any data base objects that might have been damaged during the previous system termination (if the system terminated abnormally). If data base recovery is needed, a prompt is displayed allowing the operator to specify when recovery should be performed.

9. Initialize the WCBT (work control block table) for all of CPF and establish the initial number of CPF job structures as specified by the user through the system value QTOTJOB.

10. Set up a list of pointers to CPF objects to be used to obtain data for a VSSD (virtual storage standalone dump). This includes such objects as the WCBT, system operator message queue, and job message queues.

11. Allocate a CPF job structure for the system arbiter process and initialize that process. Synchronize the start CPF process with the system arbiter process so that the system devices can be initialized by the device configuration component.

12. Allocate a CPF job structure for the user session (if this is an attended start CPF process) so that a subprocess can be initiated in the controlling subsystem when that subsystem becomes operational.

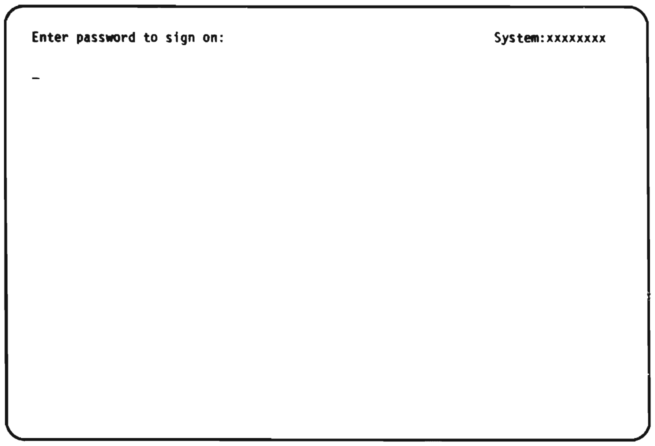
13. Initiate the start of the controlling subsystem monitor process. At this point, the STRSBS function will attempt to obtain a storage buffer of one megabyte. If that is not possible, a message (CPF0996) is sent to the QSYSOPR message queue, informing the user that storage usage has reached a critical point and must be reduced. The system is then brought up in the restricted state. No subsystem can be started during this state until storage is reduced one half megabyte beyond the one megabyte required for the buffer. If the initiation of the controlling subsystem fails, terminate the system.

14. Call the second data base recovery routine to process any damaged data base objects that are recovered after the start CPF process.

15. Produce the job logs of the start CPF process and the logs of any jobs that were active or being transferred at the time of previous system termination (if *KEEP was specified for the incomplete job logs option of the start CPF prompt).

WORK CONTROL DISPLAYS USED DURING THE START CPF PROCESS

The sign-on display is displayed during the start CPF process if a user pressed the Load or Power key and the hardware test indicates that the system console is functioning.



```
Enter password to sign on:                              System:xxxxxxxx

_
```

Sign-On Display

The user password input field is a nondisplay field; that is, the contents of the field do not appear on the display while it is being entered nor will they appear on a redisplay as the result of an error condition. The user must enter a valid password. The user password is used to determine the associated user ID (the user profile under which the functions are performed). If the verification of a user password fails, the sign-on display is redisplayed with the *invalid password - reenter password* message appearing on the message line.

Once a password had been received and verified, the user ID is first tested to determine if the user has job control authority.

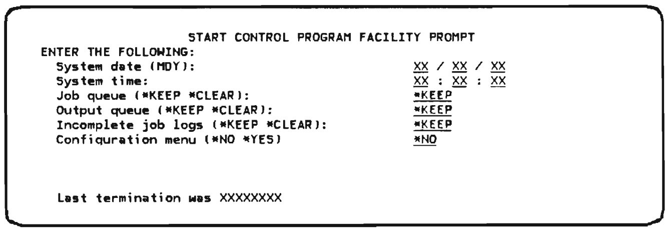
If the signed on user does not have job control authority, the sign-on display is redisplayed with the not authorized to start CPF message on the message line.

The start CPF function also verifies that the user has the authorization to use the system console. If the user does not have that authorization, the *user not authorized to console - reenter password* message is displayed on the message line of the sign-on display.

The number of attempts to enter into a start CPF session is also monitored. This number is compared to the system value QSCPFSIGN. If the number of start CPF sign-on attempts exceeds the value of the system value, the system records that fact in the history file and terminates machine processing.

After a valid user has signed on in a start CPF session, the start CPF prompt is displayed. This prompt contains the current system estimates for date and time as well as the defaults for the other input fields.

Start CPF Prompt



```
                    START CONTROL PROGRAM FACILITY PROMPT
ENTER THE FOLLOWING:
  System date (MDY):                                 XX / XX / XX
  System time:                                       XX : XX : XX
  Job queue (*KEEP *CLEAR):                          *KEEP
  Output queue (*KEEP *CLEAR):                       *KEEP
  Incomplete job logs (*KEEP *CLEAR):                *KEEP
  Configuration menu (*NO *YES)                      *NO



  Last termination was XXXXXXXX
```

where:

(MDY)	Specifies the system date format (month, day, year)
XXXXXX	Normal or abnormal to indicate the condition of the previous system termination.

The MAIN STORAGE FRAMES HAVE FAILED line is an optional line that will not appear if, during the IMPL sequence hardware tests, there were no failed main storage frames detected. If this number is greater than the value of system value QBADPGFRM, the system will be terminated unless the operator selects the configuration menu option and changes the system value QBADPGFRM to a value larger than the number of bad frames.

BASIC SYSTEM ARBITER PROCESS

The system arbiter is the central process within the CPF. It is initiated during the start CPF processing and remains active until the system is terminated. The system arbiter consists of an initiation phase program (QWCAINAR), a problem phase program (QWCAMNAR), and many event handlers.

The system arbiter is the process in which most system-wide event handlers work. Therefore, the system arbiter is mainly a collection of event handlers for many of the CPF components. Some of the components that have event handlers in the system arbiter process are:

- Device configuration–to vary and power non-peer LUDs
- Work station–device control and user interface
- Work control–timer events, system arbiter input, process initiation, and process termination
- Work monitor–system request, 5250 test request, and unsolicited data
- Switched lines–intercomponent communication concerning switched line logical unit description, control unit descriptions, and network descriptions.

SYSTEM ARBITER OVERVIEW

Figure WC-2 and the following text describe the system arbiter process.

QWCAINAR establishes the necessary event monitors for the overall system functions. The event monitor module is resolved to as a part of establishing an event monitor. QWCAINAR also initiates the logical unit services process.

Control is passed to QWCAMNAR by the machine when QWCAINAR returns to the machine. QWCAMNAR calls QDCINIT to power on and vary on devices that have been configured on the system as auto vary devices. As part of the synchronization with the start CPF function, an event is signaled back to the start CPF process.

When processing of the system devices has finished, control is returned to QWCAMNAR. QWCAMNAR then goes into a wait, waiting for a very low priority event that will never be signaled. The system arbiter is now in its operational configuration, waiting to be interrupted so it can process any of the events for which it has established a monitor.

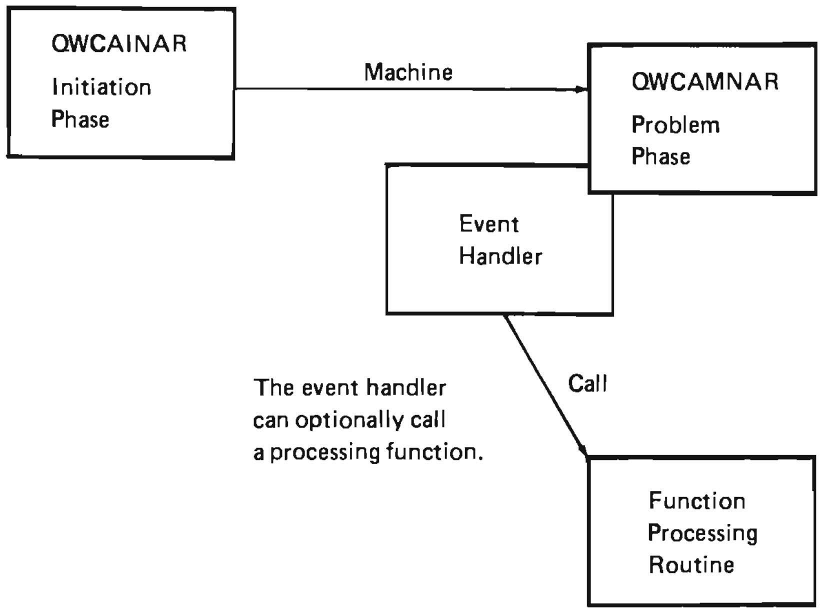


Figure WC-2. System Arbiter Process Overview

Logical Unit Services Process Overview

The logical unit services process supports peer devices. The logical unit services process in initiated by QWCAINAR during the start of the system arbiter process. Operationally, it is similar to the system arbiter in that most of the functions are provided by event handlers. The logical unit services process consists of an initiation phase program (QWCLINSR), and a problem phase program (QWCLMNSR).

Event handlers that run in the logical unit services process support LUD vary on, LUD vary off, and LUD contact events if the LUD is a peer device. An event handler in the logical unit service process changes the number of sessions allowed on a peer device. The logical unit services process also handles modes of peer devices that are not allocated by a subsystem.

START SUBSYSTEM FUNCTION

Several processes, modules, and events are involved in starting a subsystem. The start subsystem function can be invoked in one of the following ways:

- During the start CPF process
- By CSM (concurrent service monitor) while it is in the restricted state
- As a result of the requester entering the Start Subsystem (STRSBS) command

The basic configuration of the processes, modules, and events involved in the start subsystem process are shown in Figures WC-3 and WC-4.

QWCCSUUC executes in the user process, and performs some initial checks to determine if the requested start subsystem can be processed. The command processing program determines if the system is in the restricted state and (if it is) whether there is sufficient storage remaining to acquire the storage buffer that is needed by the system. If there is insufficient storage, an exception (CPF1050) is displayed to the user. The command processing program then verifies that the subsystem description exists, is not already active, is not locked by the requester or any other process, and checks to see if the subsystem description has been damaged. If processing is continued, the command processing program signals the system arbiter input event with the start subsystem subtype ID to the system arbiter process. If the start subsystem occurs during the start CPF process, QWCISCFR signals the system arbiter input event. QWCAHNAE handles the event, checks the event ID, and calls QWCASUUM. QWCASUUM performs some final checks to determine if the requested start subsystem command can be executed. If the start subsystem command has proceeded to this point, QWCASUUM obtains the system's storage buffer if it has not yet been acquired. It also checks to see if the system is in termination. If it is, the subsystem cannot be started. The message processor searches through the chain of subsystem descriptions that are chained off of the system control block to determine if a subsystem by the specified name is already active. Two subsystems with the same name cannot be active simultaneously.

If all of the checks are passed and the subsystem is not being started from the restricted state, the message processor allocates and assigns the job structure for the new subsystem monitor process. The process is then initiated and QWCASUUM waits for the machine interface to signal the process event. The status of the initiation is checked and sent to QWCCSUUC.

If a Terminate CPF (TRMCPF) or Terminate Subsystem (TRMSBS) *ALL command has been executed, the controlling subsystem is considered to be in the restricted state because all activity except for one job in the subsystem has been terminated. This case is handled separately because the subsystem monitor is already *active*.

If the subsystem is the controlling subsystem and it is being started from the restricted state, the monitor's problem phase event is signaled to the monitor that is already active. In this case, the monitor problem state program gains control immediately, rather than the initiation state program intervening. The status is then sent to QWCCSUUC by QWCASUUM.

In order to tell QWCCSUUC or QWCISCFR the results of the start subsystem request, the message processor signals the subsystem message event. The event-related data contains an indicator showing the status of the start subsystem request.

QWCCSUUC handles the subsystem message event, checks the event-related data, and then sends a message to the requester (for example, subsystem started or subsystem not started).

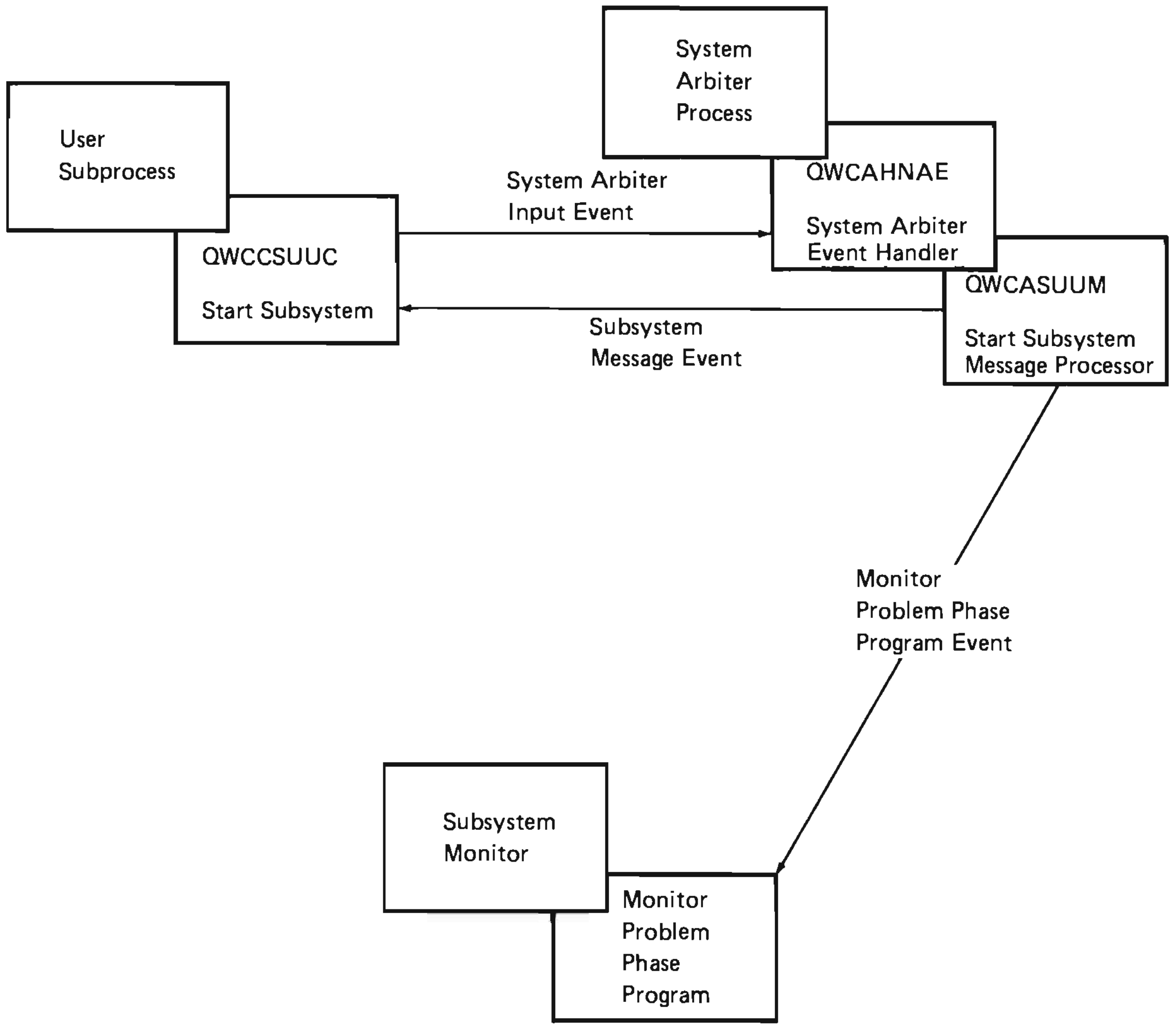

System Arbiter Process
User Subprocess
System Arbiter Input Event
QWCAHNAE
System Arbiter Event Handler
QWCCSUUC
Start Subsystem
QWCASUUM
Start Subsystem Message Processor
Subsystem Message Event
Monitor Problem Phase Program Event
Subsystem Monitor
Monitor Problem Phase Program

Figure WC-3. Start the Controlling Subsystem from the Restricted State

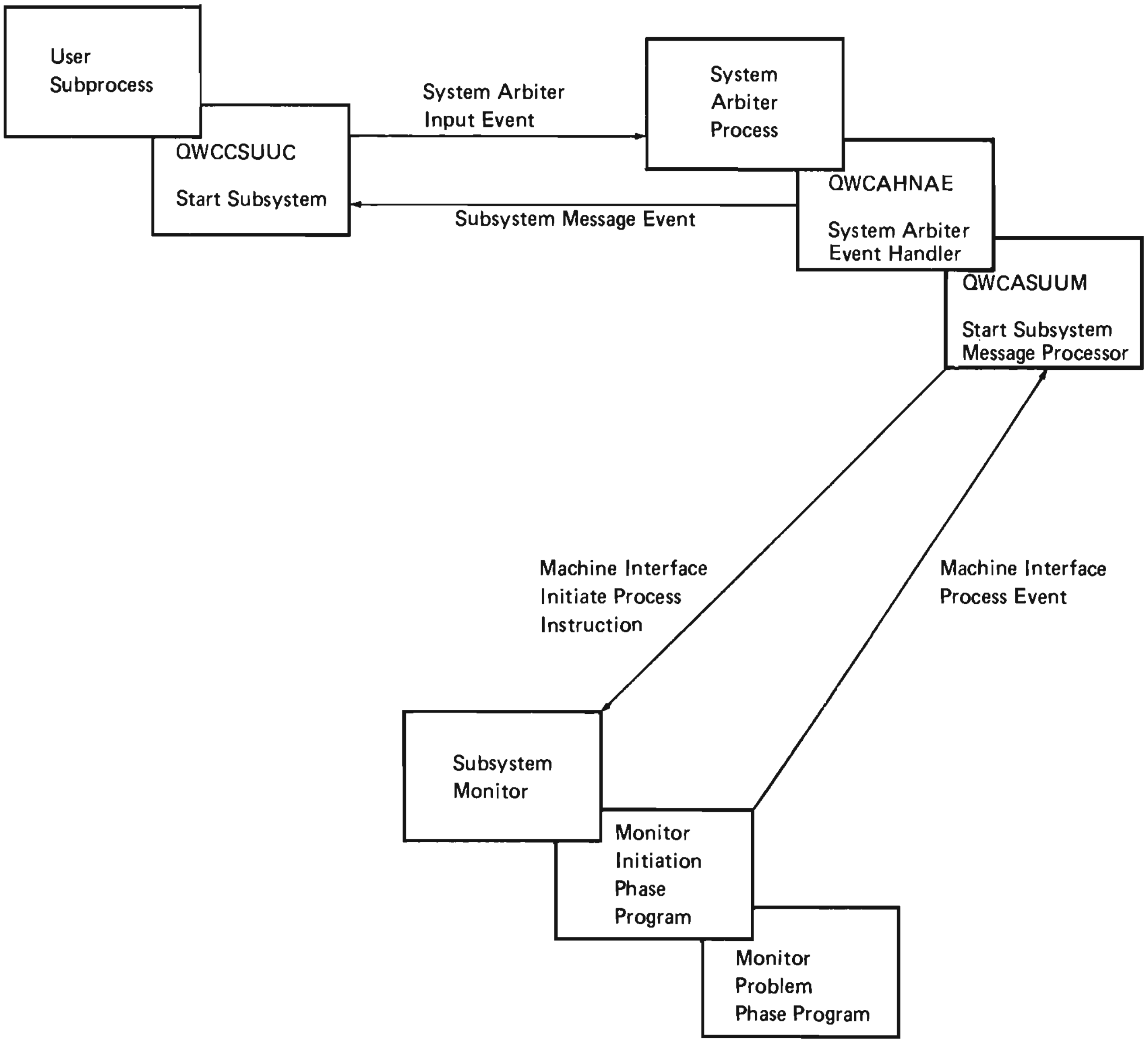

User Subprocess
QWCCSUUC
Start Subsystem
System Arbiter Input Event
System Arbiter Process
QWCAHNAE
System Arbiter Event Handler
Subsystem Message Event
QWCASUUM
Start Subsystem Message Processor
Machine Interface Initiate Process Instruction
Machine Interface Process Event
Subsystem Monitor
Monitor Initiation Phase Program
Monitor Problem Phase Program

Figure WC-4. Start Subsystem

SYSTEM/SUBSYSTEM TERMINATION FUNCTION

There are four types of CPF terminations performed by commands. They are:

- Terminate Subsystem (TRMSBS): By specifying the name of a subsystem to be terminated, only that subsystem will be terminated.
- Terminate Subsystem (TRMSBS): By specifying *ALL, all active subsystems will be terminated, except the controlling subsystem. This command leaves only the requester's session active in the controlling subsystem.
- Terminate CPF (TRMCPF): This command terminates all active subsystems except the controlling subsystem. This command leaves only the system console active in the controlling subsystem.
- Power Down System (PWRDWNSYS): This command powers down the entire system.

Note: TRMSBS *ALL and TRMCPF are valid only from interactive jobs in the controlling subsystem.

There are keywords on the terminate commands that specify the method of termination, controlled or immediate, and an allowable delay before termination begins. The delay time applies only to a controlled termination.

A controlled subsystem termination is one in which the subsystem monitor process signals an event to each subprocess in the subsystem. An event handler in each subprocess sets a flag in the WCB of the process.

An immediate subsystem termination is one in which the subsystem monitor process executes a terminate process against each subprocess in the subsystem. This does not let the user control the termination of the subprocess but the system will provide a basic cleanup that will close opened files.

Figure WC-5 and the following text describe the configuration of the processes, modules and events involved in the termination function.

1 The terminate function command processing program (QWCCSDSC or QWCCSDUC), which executes in the user process, checks to determine if the requested terminate was entered from a job from an acceptable environment. TRMSBS *ALL and termination of the controlling subsystem are allowed only from interactive jobs initiated from *SIGNON work station entries in the controlling subsystem. TRMCPF is allowed only from an interactive job running in the controlling subsystem.

2 If processing is continued, the command processing program signals the system arbiter input event to the system arbiter process.

3 QWCAHNAE handles the event and calls the appropriate terminate message processor (QWCASDSM or QWCASDUM). The terminate message processor performs the final checks to determine if the requested terminate command can be executed. If command processing continues and if a delay time was specified with a controlled termination, a timer event monitor is established to correspond to the delay time as specified on the command. If PWRDWNSYS *IMMED is specified, a timer event monitor, determined by system value QPWRDWNLMT, is set up.

4 The subsystem control event is signaled to the monitor processes involved in the termination. QWTMESBC is the event handler for the subsystem control event.

5 If a timer event was established, at the expiration of the time interval, a timer event will occur. The timer event handler (QWCAT1TE or QWCAT2TE) will be invoked in the system arbiter process.

6 For a controlled termination, the timer event handler signals the subsystem control event to the monitor processes involved in the termination (if they have not already completed the termination processing) or indicates immediate termination for PWRDWNSYS *IMMED, if the timer event occurs (machine processing is immediately terminated).

7 When a process terminates, a machine process event (signaled by the machine) is signaled to the process that initiated the terminated process. The system arbiter process is the process that initiates all of the subsystem monitor processes so it will be signaled when these processes terminate. QWCASCUE processes the event data, updates the system control block, and terminates the QLUS process and itself (the system arbiter process), if the system is being terminated, after the last subsystem monitor process has terminated.

8 When the system arbiter process terminates, a machine process event i signaled to the start CPF process. QWCITRSE processes the event data and terminates machine processing.

There are conditions, other than by command, that will cause CPF to automatically terminate the system. During start CPF these conditions range from the inability to locate a system program, to invalid user sign-on attempts in excess of the specified limit, and to system arbiter process initiation failure.

If the controlling subsystem unexpectedly terminates, the CPF automatically initiates system termination.

When the CPF does terminate the system automatically, it does not cause the machine to power itself completely down; rather, it terminates to a checkstop state. In terminating to this state, the console LED lights display four hex digits as an indication to the reason for the termination.

The machine has allocated a subset of values for the CPF to use in indication of system terminating error conditions. If the CPF terminates the system to a checkstop state, the console LEDs will display a hex 09ZZ where ZZ is the CPF termination reason code. ZZ will be a value from hex 80 through hex FF.

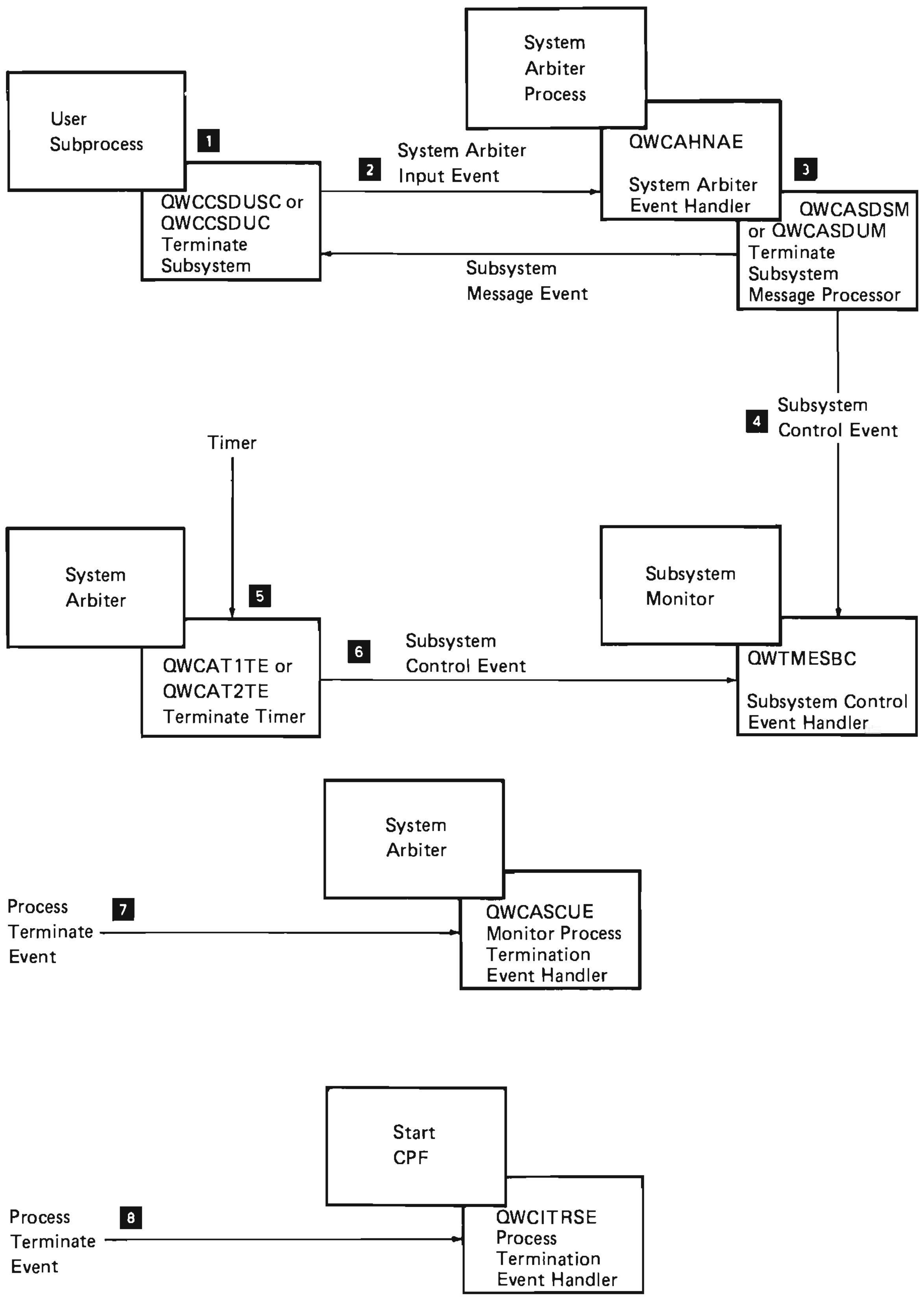


Figure WC-5. System/Subsystem Termination Overview

Allocate/Deallocate System Object

Certain types of system objects can be allocated for use by a job step through the Allocate Object (ALCOBJ) command. This allocation involves obtaining internal locks upon the objects. Allocated system objects can be explicitly deallocated through the Deallocate Object (DLCOBJ) command, or at the end of the job step the objects will be implicitly deallocated.

The work control portion of allocate object and deallocate object support consists of three modules:

- QWCSADVR–allocate/deallocate input validity checker
- QWCCALOC–allocate object command processing program
- QWCCDAOC–deallocate object command processing program

QWCSADVR is called by the command analyzer to validity check the input for the Allocate Object (ALCOBJ) and Deallocate Object (DLCOBJ) commands.

For each object specified on the command, QWCSADVR does the following:

- Check if the object type specified is valid and allocatable.
- If a library name was specified for an object, check if QSYS or *LIBL is specified if QSYS is the only valid library.
- Check if a member name was specified for an object. (Member names are allowed only for data base files.)
- Check if the specified lock state is applicable for the specified object type.

If any errors in the specifications for objects on the command are detected by QWCSADVR, it will send diagnostic messages to the program queue of its caller (the command analyzer) and then signal exception CPF0002 after completing its processing of the command input. If no errors are detected by QWCSADVR, the module returns normally.

If no errors are detected by QWCSADVR, the command analyzer transfers control to QWCCALOC for an Allocate Object command, or to QWCCDAOC for a Deallocate Object command. The processing done by these modules, up to the locking or unlocking of objects, is basically the same.

For each object specified on the command, QWCCALOC and QWCCDAOC will do the following:

1. If the object type specified is FILE and a member name is specified, resolve to the member. If the member is found, copy the file name, member name, library name, and lock state into a list of files. If the object type specified is FILE and a member name is not specified, resolve to the file. If the file is found, check the file control block to determine if the file is a data base file or a device file (device files are not allocatable). If the file is a data base file, check if the file currently has any members. If the file has members, copy the file name, library name, and lock state into the list of files. If the object type is DEVD, resolve to the device LUD. If the device type is peer, reject the command for QWCCALOC. If the device is found, copy the pointer to the LUD and the lock state to a machine lock list. Save the index of the device entry in the lock list in an array of indexes, to be used when invoking the SWHLUD macro, later.

2. If the object type requires a special object handler, call the special handler to find the object and perform any special processing on it. If the special object handler indicates successful completion, copy the pointer returned by the special object handler and the lock state into the machine lock list.

3. If the object type does not meet the conditions in items *1* or *2* above, resolve to the object. If it is found, copy the pointer and lock state into the machine interface lock list.

If any exceptions were detected during the processing of the input (such as object not found, library not found, etc.), diagnostic messages will be sent by QWCCALOC or QWCCDAOC, followed by an exception indicating that the allocation or deallocation was not performed.

If no exceptions are detected during the processing of the input and the command is:

- Allocate Object: QWCCALOC sets up the type of lock request and the length of time to wait for the lock request to be granted from what was specified on the WAIT parameter of the Allocate Object command. QWCCALOC then invokes the ?LOCK macro with the file list and the machine lock list as input. The ?LOCK macro calls the QDMLOCK module of common data management to obtain the locks on the objects.

- Deallocate Object: QWCCDAOC invokes the ?UNLOCK macro with the file list and machine lock list as input. The ?UNLOCK macro calls the QDMUNLCK module of common data management to unlock the object locks.

QWCCALOC, if the locks are successfully obtained, invokes the ?SWHLUD macro with the machine interface lock list and the array of indexes to device entries in the lock list as input. The ?SWHLUD macro performs the special processing necessary for switched line devices.

If the locks are not obtained, QWCCALOC signals an exception identifying the problem.

If the locks are not released, QWCCDAOC signals an exception identifying the problem.

DISPLAY FUNCTIONS FOR WORK CONTROL

The display functions of work control present to the requester information pertaining to the system in general, the subsystems, and jobs that are in the system and on queues. A display can be directed to either the requester display device, if interactive, or to a print file. The following displays are included:

- Display system status: This display presents a view of the operational aspects of the system as a whole. It shows the effect of the current processing load on the system. The major parts of this display are:
  - Current system date and system time
  - CPU use during the interval, as a percent of the elapsed time
  - Number of user and system jobs that are currently active in the system or on queues
  - Percentage of auxiliary storage that is currently in use and total amount of auxiliary storage
  - Percentage of the maximum possible addresses currently in use
  - Detailed performance information about each storage pool

  For each storage pool:
  - System pool ID
  - Total size in K-bytes
  - Amount of storage in K-bytes, reserved for machine functions
  - Rate, in faults per second, of page faults against data space pages and data space index pages
  - Rate, in pages per second, that data base pages are being brought into main storage
  - Rate, in faults per second, of page faults against nondata base pages
  - Rate, in pages per second, that nondata base pages are brought into main storage
  - Maximum activity level for the pool
  - Rate, in transitions per minute, of transitions from an active to a waiting state
  - Rate, in transitions per minute, of transitions from a waiting to an ineligible state
  - Rate, in transitions per minute, of transactions from an active state to an ineligible state

- Display system: This display shows which subsystems are active, the status of each subsystem, and the load being processed within each subsystem.

- Display subsystem: This display presents all of the jobs that are active in the system or residing on spooling job and output queues.

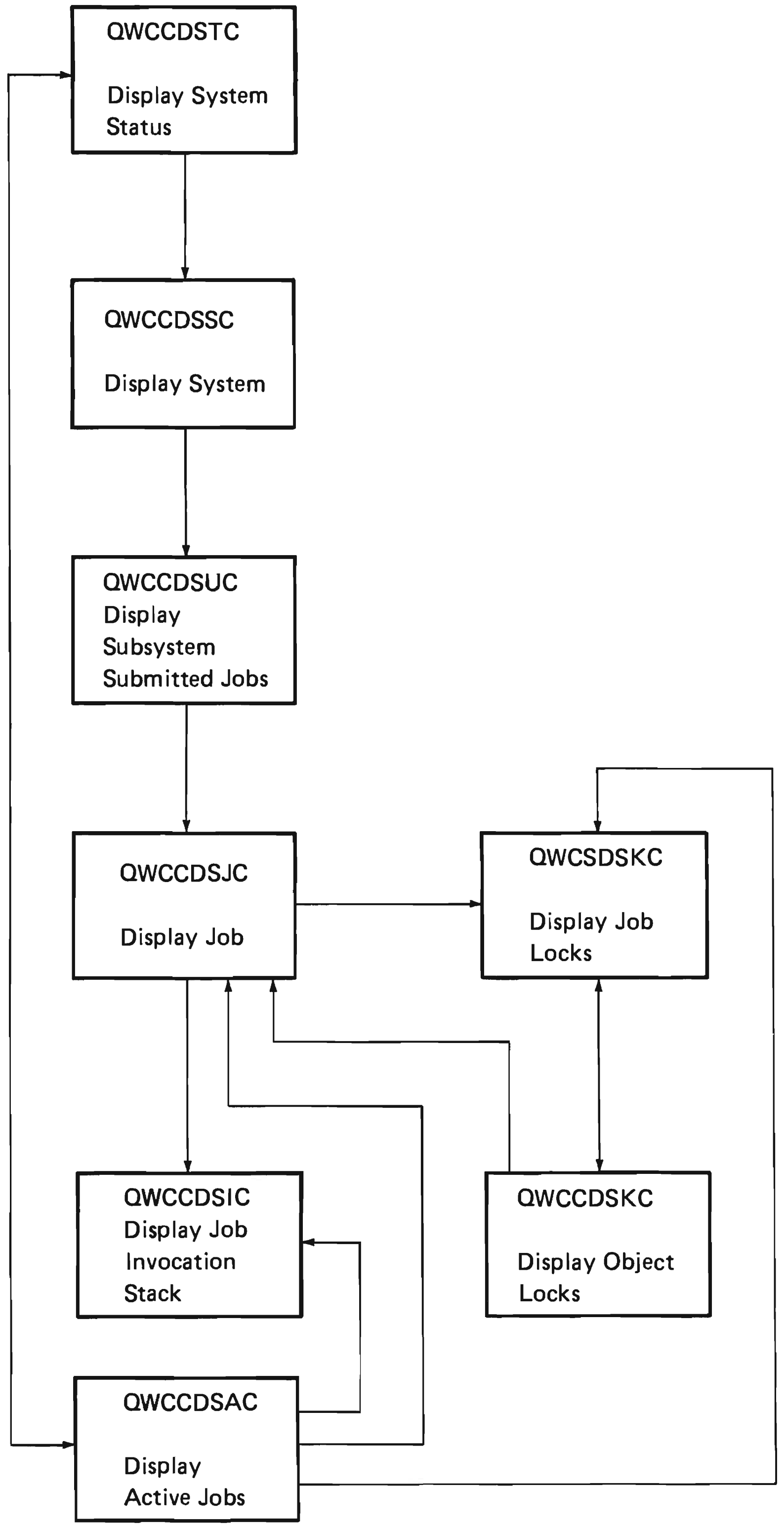
- Display submitted job: This display presents a list of:
  - All jobs submitted by this job, by this user, or at this work station.

- Display job: This display presents a detailed display of a particular user job that is active or on queues. The following is presented for each job:
  - Job status attributes
  - Job definition attributes
  - Job execution attributes (if active)
  - Program invocation stack (if active)
  - Spooled files
  - Locks (if active)
  - Commitment control status (if active)
  - Library list (if active)
  - Open files (if active)
  - File overrides (if active)
  - Job log (if active or on job queue)

- Display object locks: This display presents all object locks in the system for a specified object, including held locks and locks being waited for.

- Display active jobs: This display presents performance and problem information for the active jobs in the system. The number of active jobs and the CPU use during the interval are also presented.

The following diagram shows the module relationship of the work control display modules:



Work control is also responsible for data area, system value, class, and network attribute displays.

CLASSES

A class is an object that contains a set of resource management information. This information controls the operational characteristics of the process that supports the routing step. These characteristics include:

- Timeslice: The length of time during which the job step is allowed to process when selected for execution.
- Machine execution priority: A selection criterion.
- Purge option: Indicates whether the process and its associated structures should be paged out of main storage during a long wait or at timeslice end.
- Default wait time-out: The amount of time that an instruction, which goes into a wait, remains in that wait if the instruction specifies that the default is to be used.

A class is associated with each routing entry in a subsystem description by the Add Routing Entry (ADDRTGE) command or Change Routing Entry (CHGRTGE) command. When a job is routed through the entry, the subsystem monitor copies the values contained in the class into the PDT (process definition template) for the initiation of the process that supports the job step.

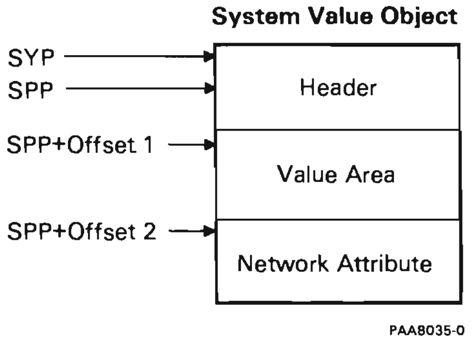
Supported operations on a class are create, delete and display. The delete operation is supported by the librarian's delete object command processing program. Operations on a class are internally serialized through the use of locks to avoid inconsistencies or contention due to concurrent operations.

SYSTEM VALUE FUNCTIONS

System values are IBM-defined entities that are used to control system actions and to provide information to the user. The user can change the value of most system values to tell the system what to do in case a particular situation arises, such as what to do if the system console cannot be brought up during the start CPF process. The user can also change other system values that determine such things as the format of the system date, the date separator character, the system time, and the format in which numeric data is to be presented. Certain system values cannot be changed by the user, but are to be used only to display information or to use in if-else clauses in user CL procedures.

QWMSYSVAL is installed with the system and is a permanent object. It contains the names of all of the system values and their current values as well as the values for all of the Network Attributes. The system value object consists of two parts.

- Header: This consists of entries of equal length for each system value. Each entry contains the name of the system value, the displacement to the value's entry in the value area and a special code indicating special type characteristics of the value.
- Value area: This area contains the value for each system value. The displacement to a particular value's entry is in the header entry.
- Network Attribute value area: This area contains the value for each network attribute.



The user is allowed to perform the following functions with system values:

- QWCCDSLC–Display System Value: This module presents to the requester the name of the value requested and its current value. The display can be directed to either the interactive device of the requester or to the job default spooling queue.
- QWCSRTLR–Retrieve System Value: This function can be invoked in one of two ways: work control provides a macro interface by which the type, length, and value of the system value can be retrieved in an area provided by the requester or through the Retrieve System Value (RTVSYSVAL) command in a CL program, in which the value is retrieved into a CL variable provided by the requester. The type and length of the CL variable are checked before the value is retrieved to verify that they are the same as the type and length of the system value.

- QWCCCHLC–Change System Value: This module changes the value of the requested system value, provided that the value can be changed by the user, and the new value is of the correct type and length for the requested value. Some values can take on only a limited number of different values, so these are also checked for in those particular cases. If the system date or system time values are changed, the actual system clock is changed. The processing for some system values involves more than just the command processing program. As a result, QWCAUPRM executes in the system arbiter process that handles these cases.

- QWCAUPRM–Update Processor Rollover Message: This module gains control in the following manner: QWCCCHLC sets up event related data and then signals the system arbiter input event with the event subtype indicating that QWCAUPRM is to be called. QWCAHNAE gains control in the system arbiter process, and calls QWCAUPRM, passing the event data. Depending on the system value being changed, event monitors can be canceled and reestablished with new compare values, the system clock can be changed, addressability to a new prototype NRL (name resolution list) created by QWCCCHLC can be established, and values in the WCBT can be updated. QWCCCHLC waits for a response from QWCAUPRM.

- QWCSCRLR–Create/Convert System Value Object Routine: This module converts the system value object to a new release/modification level following installation of CPF. It adds new system values without affecting any changes made by the user to existing system values. QWCSCRLR also re-creates the system value object if it becomes damaged or destroyed.

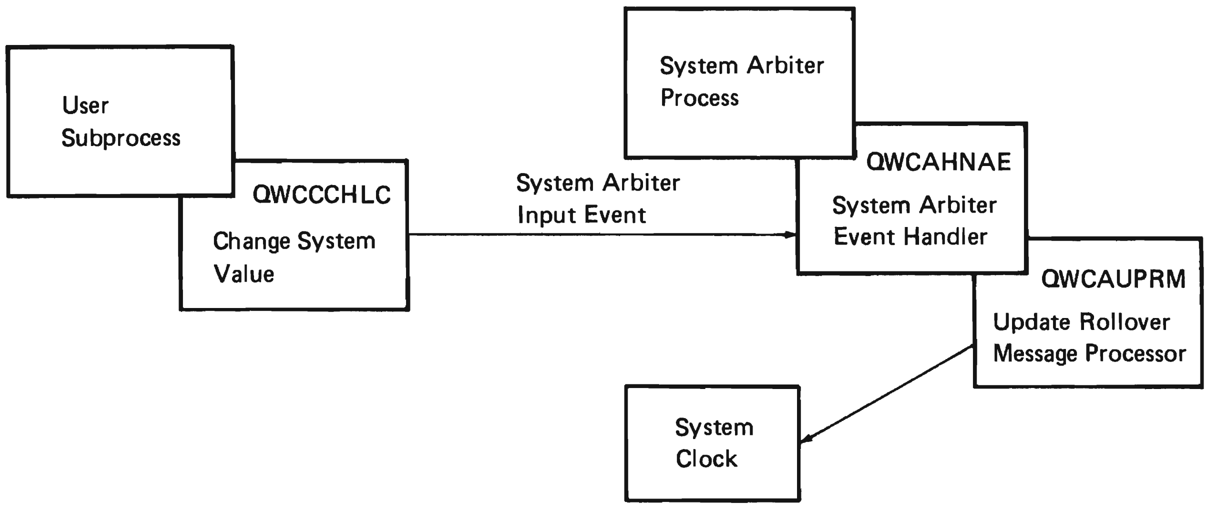


Figure WC-6. Module Interrelations for Changing a Date or Time on the System

Network Attributes Support

Network attributes describe characteristics of a system that may exist in a network of systems. For example, the system name, which is a unique identifier of a system within a network, is a network attribute.

The user is allowed to perform the following functions within network attributes:

- QWCCDSNC–Display Network Attributes: This module displays the names of all the network attributes and their current values. The display can be directed to either the interactive device of the requestor or to the job default spooling queue.
- QWCCCHNC–Change Network Attributes: This module changes the network attributes. Some attributes require certain values, or cannot be changed in certain environments. If all of the values are valid, the requested changes are made; if any of the values are invalid, no requested changes are made.

DATA AREAS

A data area is an object that can be used to store and retrieve information. It is most commonly used to contain information that the user wants to communicate between executing programs.

Create Data Area–QWCCCRVC: The user can create a data area through the Create Data Area (CRTDTAARA) command.

Change Data Area–QWCCCHVC: A data area or a substring of a data area can be changed through the Change Data Area (CHGDTAARA) command.

Display Data Area–QWCCDSVC: The attributes and value of a data area can be displayed through the Display Data Area (DSPDTAARA) command.

Retrieve Data Area Pointer–QWCSRTVR (RTVSVAR macro): An external interface to the retrieve data area pointer routine (QWCSRTVR) is provided through the ?RTVSVAR macro. The ?RTVSVAR macro will return to the invoker the system pointer to the data area object (other than local data areas and group data areas).

There are also data area commands in the CL component: Send Data Area (SNDDTAARA), Receive Data Area (RCVDTAARA), Retrieve Data Area (RTVDTAARA), and Declare Data Area (DCLDTAARA).

SIGN-OFF FUNCTION

The sign-off function terminates an interactive job or group of interactive jobs (group jobs). The sign-off function can be obtained by entering the Sign-Off (SIGNOFF) command interactively or in a control program procedure. It is also used internally by the concurrent service monitor component, through the sign-off macro interface.

On the Sign-Off command, the user can specify the dial-up line disposition of the device that is being signed off, through the use of the DROP keyword. (The keyword is ignored if the device is not a dial-up device.) The user can request that the dial-up line be dropped, that the dial-up line connection be retained, or that the dial-up line disposition parameter, which is specified in the device description of the work station, be used as the default.

The LOG keyword on the Sign-Off command determines if the job log is to be spooled or deleted. If *NOLIST is specified or defaulted, the ?CHGJMQ macro is executed, which sets indicators in the JMQ so that no job log file is created in the termination phase program. If *LIST is specified, the job log is spooled.

The sign-off command processing program is called by the sign-off macro interface or by the command analyzer if the user enters the Sign-Off command. The sign-off command processing program invokes the work monitor terminate process macro and issues a Terminate Process Machine Instruction instruction. The termination code is set to indicate an abnormal signoff if an exception exists or there is an event in the invocation stack. Otherwise, the termination code is set to indicate a normal sign-off.

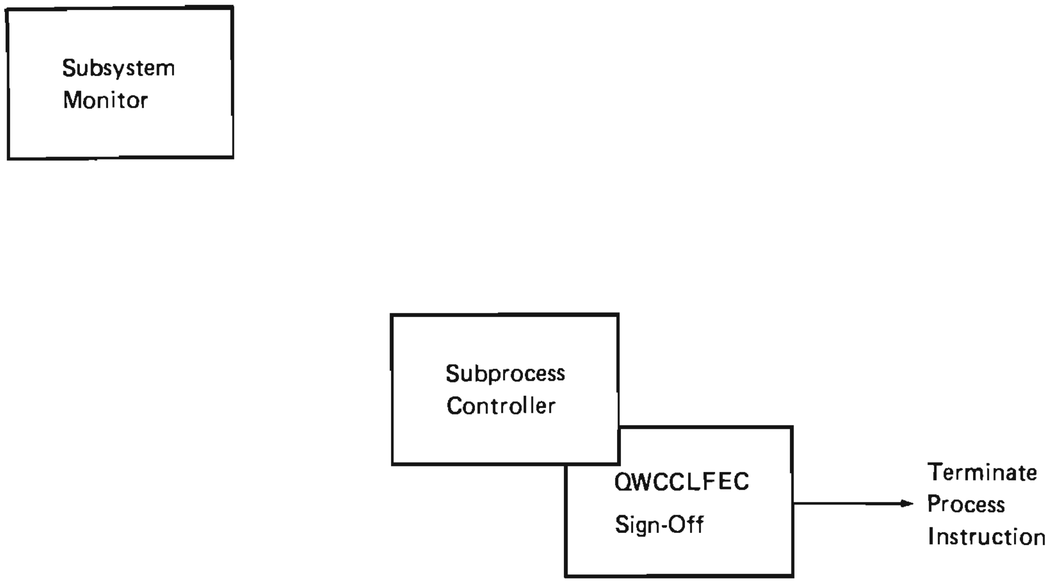


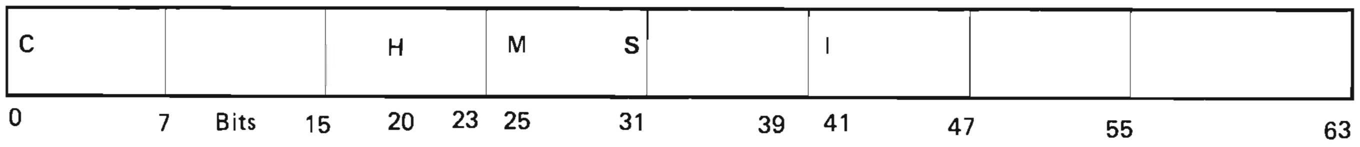
Figure WC-7. Sign-Off Overview

SYSTEM DATE/TIME SUPPORT

The work control component provides the system date/time function for the CPF. The system clock is set and maintained based on the date/time algorithms established for CPF. The understanding of the handling of time is based on knowing how the system time is setup. The following describes how time is kept in the system.

The system clock is an 8-byte counter that is incremented at a rate equivalent to 1024 microseconds at bit position 41. Even though this is the bit position that is incremented in the system, this does not have to be so, as long as the time interval at bit 41 is equal to 1024 microseconds. CPF does not work with bits in the system clock after bit 41 (bits 42 through 63).

The basic layout of the clock is shown with some of the important bit positions designated:



where:

C = Century bit
H = Approximate hour bit
M = Approximate minute bit
S = Approximate second bit
I = Clock increment bit

System/38 Clock Layout

With the clock being stepped at the specified rate, the total time interval that can be contained in the clock (before it rolls over) is about 142 + years. CPF has established the system time such that at the start of the year 2000 the clock will contain only the century bit set (8000 0000 00-- ---X; --- indicates unused so that it can not be stated exactly what these bits may contain). By picking the clock setting for the start of the century, this leads to an epoch on around August 27, 1928.

This implementation of the system time combined with the duration of the clock, leads to the condition where some years are redundant. An arbitrary rule is set up to remove this redundancy, where a test is made such that a year value equal to or greater than 40 results in a year from 1940 through 1999. A year value of less than 40 results in a year value from 2000 through 2039.

System Timer Support

Work control maintains a system date and time. At the start CPF process, an estimate of the date and time is made, based on whether the last termination was normal or abnormal.

If the system was terminated normally the system's estimate of the date and time is the date following that on which the system was terminated at zero time (00:00:00). For example, if the termination occurred on 07/13/80 at 18:30:00, the date and time presented to the user at the next start, if attended, is 07/14/80 at 00:00:00.

If the system was terminated abnormally, the system's estimate of the date and time is the date and time at which termination occurred, plus a small amount of time to ensure that there is no overlap with the system time prior to termination.

In order to support these estimates and to increment the date system value (QDATE) when necessary, the system arbiter establishes an event monitor for a 24-hour rollover event. The handler for this event is QWCAROTE (24-hour).

The rollover handling maintains the date system value and saves the system timer value for the start of the current day and the start of the next day. At each 24 hour interval and at system start, a timer event is established that will expire at the end of the next 24 hour interval. The start of these intervals corresponds to zero hours according to the timer epoch fixed by CPF. The saved time-of-day clock value for the next occurrence of zero hours is used at the next system start to compute the estimated date and time.

STORAGE POOL/MPL RESOURCE MANAGEMENT

The following explanation of CPF storage pool/MPL resource management deals basically with the system arbiter functions. For a discussion of the subsystem monitor functions in CPF system resource management, see *Work Monitor*.

The machine defines 16 storage pools and MPL (multiprogramming level) classes, numbered from 1 through 16. Storage pools are used to partition main storage into discrete and independent areas, in which processes can run and contend for storage only with processes running in the same storage pool. MPL classes are used to set the number of processes that can concurrently be eligible for execution in that MPL class. These storage pools and MPL classes are used in a one-to-one correspondence by CPF. Each storage pool in the system is said to have an activity level associated with it. This storage pool activity level is the multi-programming level of the associated MPL class.

Machine storage pools 1 and 2 are defined by the CPF to be the machine pool and *BASE pool, respectively. These pools will always be allocated while CPF is in normal operation. The attempted minimum size of the machine pool is set to a system value, QMCHPOOL. Non-nucleus machine code will run in the machine pool. The attempted minimum size of the *BASE pool is set to the system value, QBASPOOL. The system arbiter process runs in the *BASE pool. Pool descriptions in subsystem descriptions can be set to the *BASE pool, so an individual pool is not allocated and processes will be initiated into the *BASE pool by the subsystem monitor.

The apportionment of storage, storage pools, and MPL is controlled by QWCARQRM. Subsystem monitors make requests for resources (storage, storage pools, MPL) by signaling the system arbiter input event with the request resources subtype. QWCAHNAE will be invoked and will call QWCARQRM with a pointer to the event related data from the system arbiter input event. The event related data consists of a pointer to a request resources message space.

This message space contains information about the monitor that signaled the system arbiter input event and a number of requests for resources. Each request in the message consists of a request type, a subsystem description storage pool ID, machine storage pool ID, a requested MPL, a requested storage pool size, and a status of request indicator.

A pool reduction or deallocation causes a machine exception. If a machine exception is signaled, the reserved storage released event monitor is enabled, a message is sent to the system operator message queue telling which system pool could not be reduced, and the pool request is stored in the resource management control block.

Before processing any of the requests, QWCARQRM first determines the amount of storage required for machine interface pool 1 (machine pool) and machine interface pool 2 (*BASE pool) and satisfies their requirements first (if there is sufficient storage available to do so). Any nonallocated storage is then available to satisfy monitor requests. After all the requests are processed, any remaining storage is allocated to the *BASE pool.

The individual requests are processed sequentially by QWCARQRM and if sufficient resources are available, each request will be granted in turn and the status of the request will be set to request granted. If the request cannot be granted, the requested MPL and requested storage pool size will be set to whatever was actually obtained for the request, and the status of the request will be set to an appropriate failure condition.

When a subsystem is started, storage pools (and MPL classes) can be allocated for the subsystem, depending upon the pool descriptions in the subsystem description and the number of storage pools and amount of storage available for allocation (see Figure WC-8).

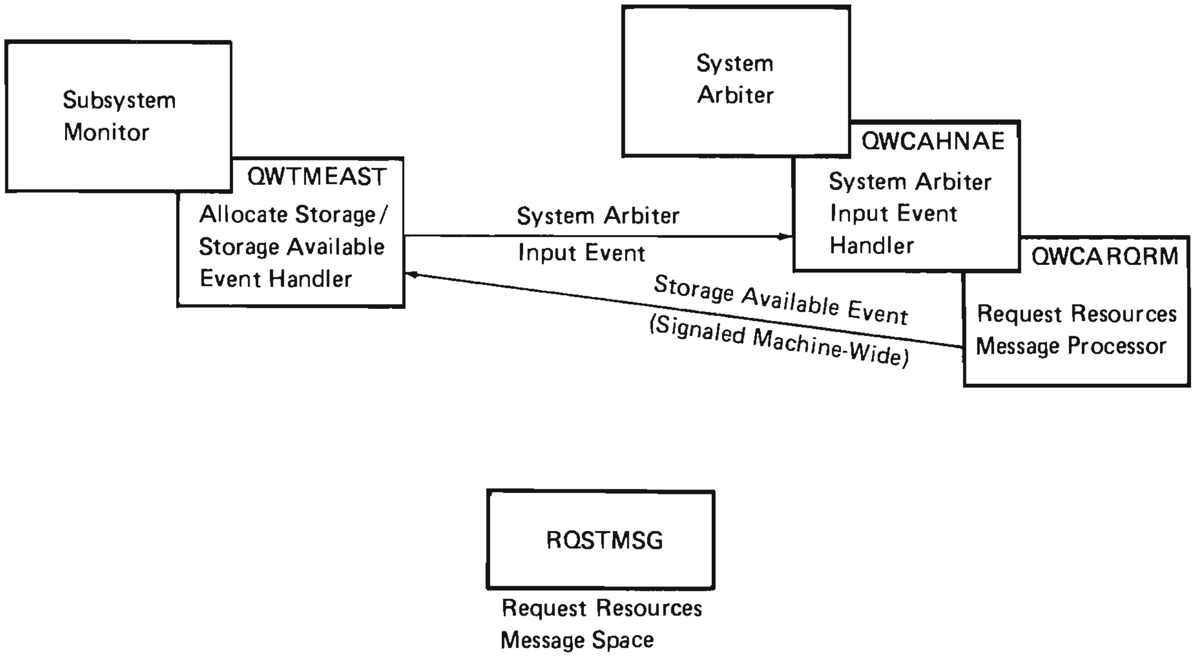

Subsystem Monitor
QWTMEAST
Allocate Storage/ Storage Available Event Handler
System Arbiter
QWCAHNAE
System Arbiter Input Event Handler
System Arbiter Input Event
Storage Available Event (Signaled Machine-Wide)
QWCARQRM
Request Resources Message Processor
RQSTMSG
Request Resources Message Space

Figure WC-8. Monitor Start-Up, Outstanding Requests, Change Subsystem Description (Active SBS) and Initialize Pool and Increase Size of Pool Request

If the monitor PCS pointer in the header portion of the request resources message space is null, the requesting monitor does not need to have a storage allocated event signaled to it. This is true if the monitor is making MPL-only changes, decreasing pool sizes, or deallocating pools. These requests are always granted, so the monitor does not need a status of request response. In this case, QWCARQRM will destroy the request resources message space at the end of its processing. These types of requests are possible through a Change Subsystem Description command, executed against an active subsystem (see Figure WC-9), or the termination of the last active process in a storage pool that was set to a size of zero by a previous Change Subsystem Description command, executed against an active subsystem (see Figure WC-10).

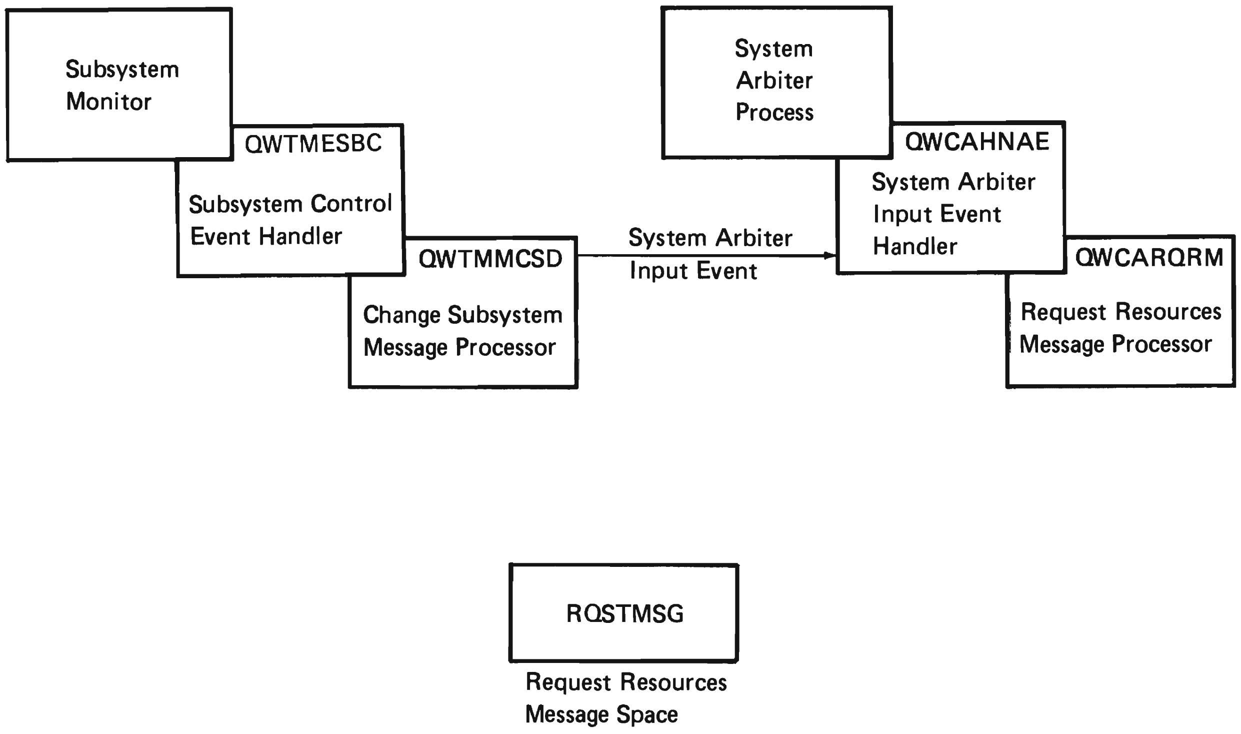

Subsystem Monitor
QWTMESBC
Subsystem Control Event Handler
QWTMMCSD
Change Subsystem Message Processor
System Arbiter Input Event
System Arbiter Process
QWCAHNAE
System Arbiter Input Event Handler
QWCARQRM
Request Resources Message Processor
RQSTMSG
Request Resources Message Space

Figure WC-9. Change Subsystem Description (Active SBS), MPL-Only, Decrease Pool Size, and Deallocate Pool (Only if No Processes Active in the Pool) Requests

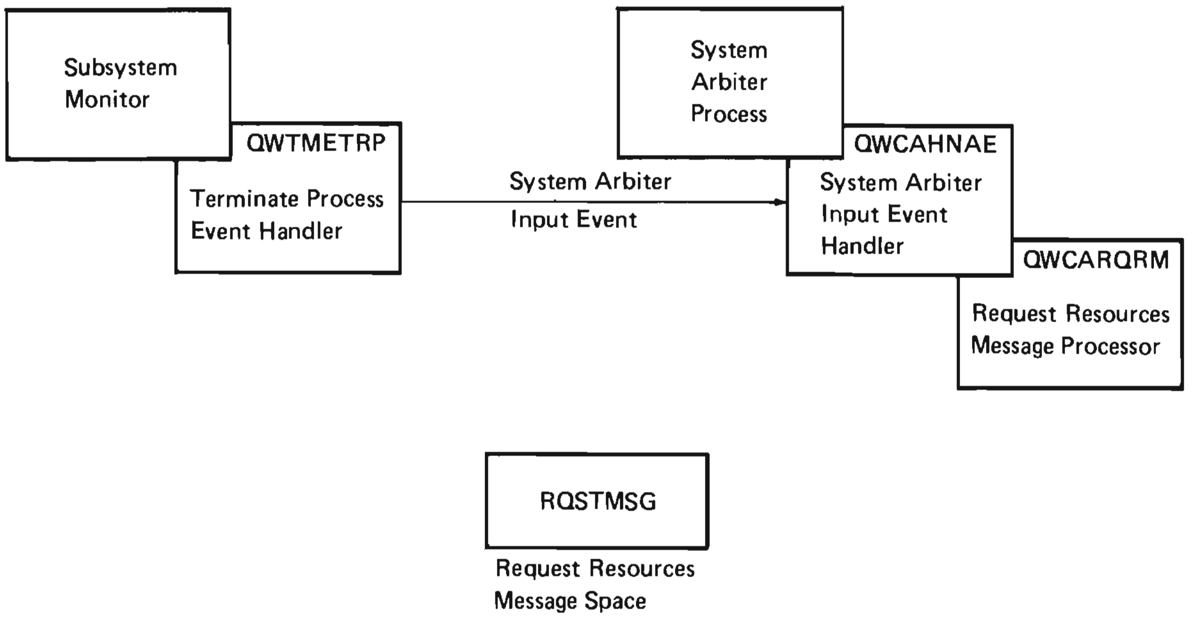

Subsystem Monitor
QWTMETRP
Terminate Process Event Handler
System Arbiter Input Event
System Arbiter Process
QWCAHNAE
System Arbiter Input Event Handler
QWCARQRM
Request Resources Message Processor
RQSTMSG
Request Resources Message Space

Figure WC-10. Termination of the Last Process Running in a Deallocated Pool

If the monitor PCS pointer in the request resources message space is not null, the storage allocated event is signaled to the requesting monitor, with the pointer to the request resources message space as event related data (see Figure WC-11). This is the case whenever an initialize pool or increase pool size request is made.

If there is not enough storage available to grant all the requests of a monitor after the monitor has received its response from the system arbiter, it monitors for a storage available event (see Figure WC-8).

If there is any storage available at the end of processing an entire message, QWCARQRM signals a machine-wide storage available event to any monitors that are listening for it (see Figure WC-8). If the storage available is greater than or equal to the minimum amount required to initalize a storage pool and at least one storage pool is available for initialization, the storage available event is signaled with a compare value of one. Otherwise, the event is signaled without a compare value.

When reserved storage is released, the reserved storage released event is signaled, and QWCARSRE is invoked. QWCARSRE retrieves the event data that has the pool ID. QWCARSRE then checks to see if there is a pending reduction or deallocation for the pool. If so, QWCARSRE reduces or deallocates the pool, places extra storage in the *BASE pool, and signals the storage available event.

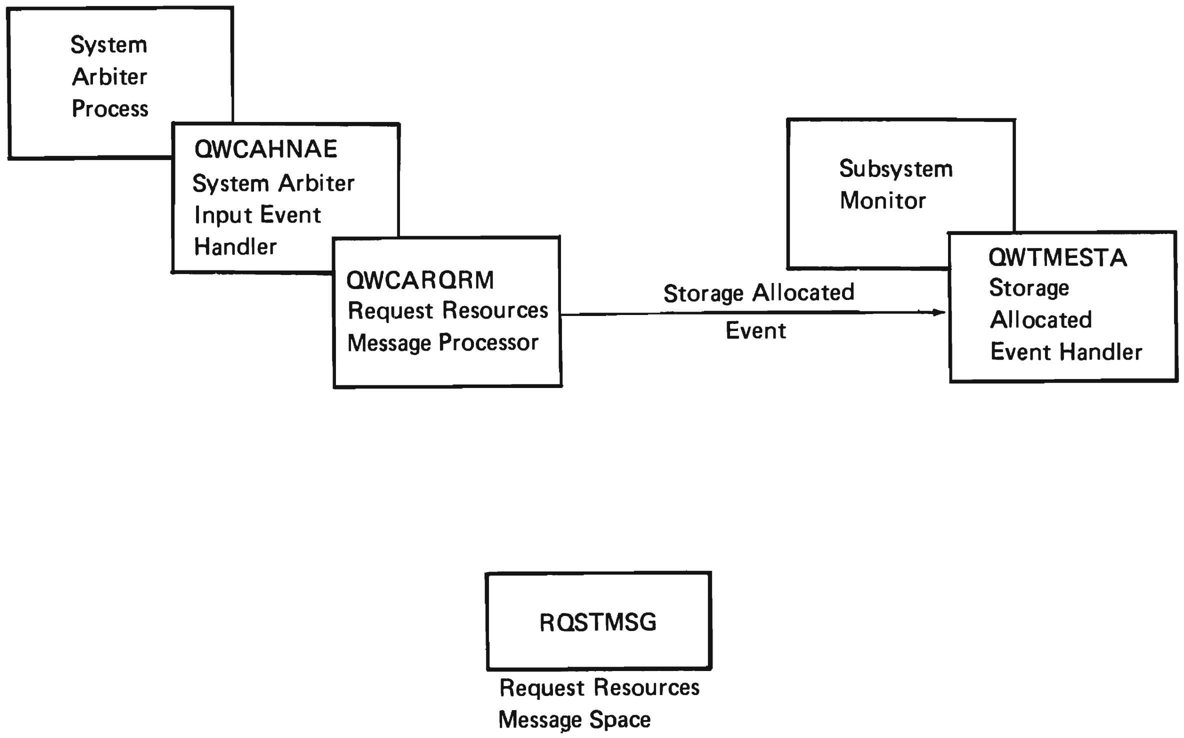

System Arbiter Process
QWCAHNAE
System Arbiter Input Event Handler
QWCARQRM
Request Resources Message Processor
Storage Allocated Event
Subsystem Monitor
QWTMESTA
Storage Allocated Event Handler
RQSTMSG
Request Resources Message Space

Figure WC-11. Response to the SBS Monitor From the System Arbiter

WORK CONTROL BLOCK TABLE MAINTENANCE

The WCBT (work control block table) is one of the primary objects to the operation of CPF. It is a permanent space in which entries are initialized at system start. When a job enters the system, a WCBT entry is assigned to the job and it is released from the job when the job leaves the system. Available entries are selected from the WCBTEQ, (the WCBT entry queue, which is a temporary queue associated with the WCBT). When the system terminates, the queue is automatically destroyed. In the start CPF process, the table is compressed and the queue is rebuilt.

During system operation, if an attempt is made to assign an entry (a job has entered the system) but none are available, an event is signaled to the system arbiter (the system arbiter input event with WCBT subtype) to indicate that the table requires expansion. This event is handled by QWCAHNAE which in turn calls QWCAHNTM. QWCAHNTM then performs the necessary operations (initialize new entries, extend the space if necessary, and add entries to the WCBTEQ). This event can be signaled from a subsystem monitor (in the case of an interactive job or an auto-start job) or from the spool reader (in the case of a batch job). Figure WC-12 shows a subsystem monitor trying to assign an entry for a job when there are no entries available.

Note: The table handling function is complete before another process can signal the WCBT event, although QWCAHNTM has not necessarily returned.

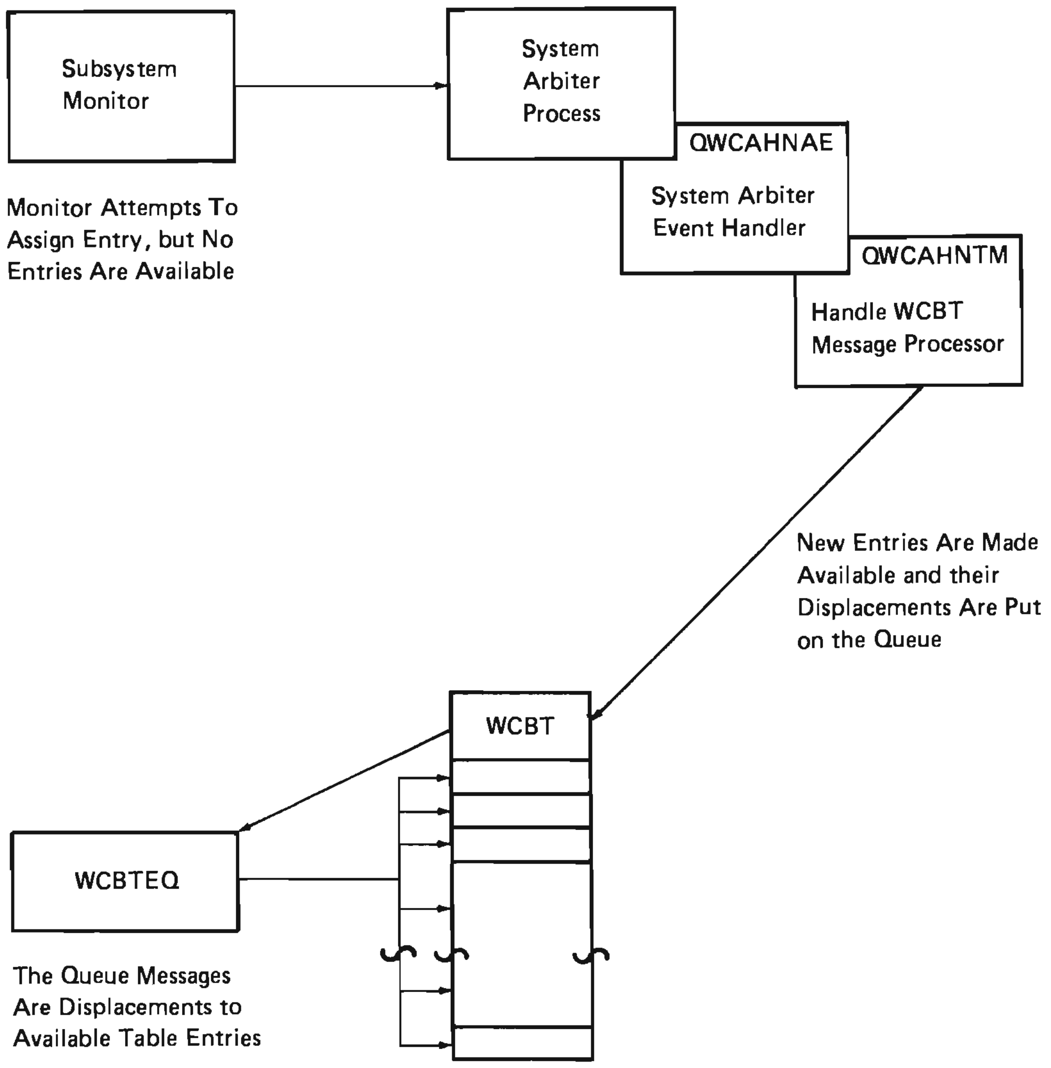

Subsystem Monitor
Monitor Attempts To Assign Entry, but No Entries Are Available
System Arbiter Process
QWCAHNAE
System Arbiter Event Handler
QWCAHNTM
Handle WCBT Message Processor
New Entries Are Made Available and their Displacements Are Put on the Queue
WCBT
WCBTEQ
The Queue Messages Are Displacements to Available Table Entries

Figure WC-12. Subsystem Monitor Trying to Assign a Job Entry with No Entries Available

MACHINE EVENT HANDLING

QWCAMCKE is an event handler designed to provide system support for the processing of the machine status, and some process events. The system arbiter process provides event monitors for the following events that QWCAMCKE handles:

Machine Status Events:

- Hex 000D 0101–Machine check
- Hex 000D 0201–Utility power failure
- Hex 000D 0202–Utility power restored
- Hex 00D0203–Utility power restored (BASE UPS)

CSNAP Timer Event:

- Hex 0014 0101–CSNAP absolute time of day

Process Events:

- Hex 0010 0701–Maximum CPU time for a process was exceeded
- Hex 0010 0801–Maximum temporary storage for the process was exceeded

In the processing of the machine status events (hex 000D), QWCAMCKE sends a message to the system operator message queue and the history file. If the event was a machine check, the event data is saved in the SCPF data object.

If the event was a utility power failure, an indicator is set in the system control block, and the system value QUPSMSGQ is inspected for a message queue name. Further checks are made as to the type, allocation, and delivery mode of the message queue. An attempt is made to send to the message queue a message informing of the power failure. If send fails for any reason or if the checks indicate the message queue is not properly set up or the system value was *NONE, the system is terminated to power off with messages sent to the system operator indicating why.

If the event was a utility power restored, an indicator is reset in the system control block, and the system value QUPSMSGQ is inspected for a message queue name. An attempt is made to send a message to the message queue, and the system operator is informed of the event and send failure (if occurred) by messages.

The CSNAP absolute time of day event is handled by issuing an MI diagnose instruction with the CSNAP op code and then re-creating the timer event for 24 hours later. The new CSNAP time is saved in the system control block.

The process events are handled by terminating the process that caused the event to be signaled (the PCS pointer for this is in the event data). The terminate process macro is invoked with the TRMCODE parameter set to indicate the type of termination (maximum CPU or temporary storage). The macro generates the code that does the actual Terminate Process instruction. QWTPITPP checks the termination code and sends the appropriate message to the job's job log (to explain why the routing step was canceled).

QWCARSRE is an event handler designed to provide system support for the processing of the reserved storage released machine event. The system arbiter process provides the event monitor for the following event that QWCARSRE handles:

Machine Resource Event:

- Hex 000C 0601–Reserved storage released

The reserved storage released event is handled by returning the released storage to machine pool when needed in the machine pool and all unneeded storage to the base pool. A storage available event is then signaled machine wide if base pool size is greater than the required minimum.

QWCATARE is an event handler designed to provide system support for the processing of the machine resource and resource timer events. The system arbiter process provides event monitors for the following events that QWCATARE handles:

Machine Resource Events:

- Hex 000C 0201–Machine auxiliary storage limit reached
- Hex 000C 0301–Machine ineligible state threshold
- Hex 000C 0401–MPL class ineligible state threshold

Timer Events:

- Hex 0014 0301–Auxiliary storage limit reached, repeating notification timer

There is no action that the CPF can take to reduce the system load that caused a machine resource event (event IDs that start with 000C). The system operator may be able to help the condition by canceling some jobs, terminating a subsystem, or changing the operating characteristics of one or more subsystems. Therefore, the only action taken in response to a machine resource event is to send a message to the system operator and the history file. In the case of a machine auxiliary storage limit reached event, a repeating timer is established to send a message to the system operator every 60 minutes until the auxiliary storage usage is reduced below the limit.

RECLAIM RESOURCES FUNCTION

The reclaim resources function is available through the Reclaim Resources (RCLRSC) command, which interfaces to machine support. Machine support enables the static storage of a process (routing) step to be reset to any desired level. This function is used by compilers, where large amounts of static storage are used by non-reentrant programs. The amount of static storage could grow in an unbounded manner if not released at natural breakpoints in a process's execution. The reclaim resources function reclaims the static storage for use by the process, and thereby improves the performance of the process. The Reclaim Resources command reclaims the static storage associated with all activations after the level is designated on the command. The reclaimed static storage is then available for use by subsequent activations.

The command processing program invokes the ?RCLPSSA macro that will chain through the PSSA and deactivate all activations with no associated invocations that have an activation mark greater than the level specified on the macro invocation. The message handler QCL invokes the macro after every call to user or CPF program.

Subsystem Description

INTRODUCTION

The SBSD (subsystem description) component of the CPF (control program facility) is used to create an object called a subsystem description. The subsystem description is used by the work monitor component to create the operational environment known as a subsystem. A subsystem description describes the environment in which a user's work is processed. Several subsystems can be concurrently active, each managing its own separate operational environment and controlling the processing of its work.

The SBSD is a standard composite object containing (1) composite object structure, (2) definitional information, (3) recovery information, and (4) working space for work control and work monitor. The definitional information consists of three parts:

- Subsystem attributes
  - Storage pool descriptions (sizes and activity levels)
  - Maximum number of jobs that can concurrently run in a subsystem
- Routing entries
- Work entries

Subsystem Attributes

The subsystem attributes are supplied through the Create Subsystem Description (CRTSBSD) command and can be modified by the Change Subsystem Description (CHGSBSD) command. A subsystem description can also be deleted using the Delete Subsystem Description (DLTSBSD) command.

Storage Pool Descriptions

A storage pool is a logical division of main storage. Routing steps running in a storage pool compete for main storage space only within their logical division of storage. Therefore, contention between programs for main storage resources can be reduced through the use of storage pools.

The attributes of a storage pool are its size and activity level. An activity level defines how many jobs can be processed concurrently within the storage pool.

In addition to storage pools that are user-defined in subsystem descriptions, the *BASE storage pool can be shared across subsystems. The *BASE storage pool is defined through system values: the size by QBASPOOL and the activity level by QBASACTLVL.

Routing Entries

A routing entry defines how a routing step is to be initiated for a job. A routing entry contains the following:

- Routing entry sequence number
- Routing data compare value
- The name of the application program to invoke for this entry
- The name of the class used for the routing step
- The maximum number of routing steps that can concurrently be active for this entry
- The identifier of the storage pool in which the routing step is to be initiated

Work Entries

Work entries are used to identify the sources for jobs that are processed in a subsystem. There are four types of work entries:

- Autostart job—The job is automatically started when the subsystem is started.
- Job queue—The job to be processed is taken from the specified job queue.

- Work station–The job is processed when a work station user signs on or when a user transfers from one subsystem to another subsystem.

- Communications–The job is started as a result of an attach (evoke) request.

The descriptions of these work entries (which are included in the subsystem description) are:

- For an autostarted job entry:
  - Job name
  - Job description name

- For a job queue entry:
  - Job queue name
  - Maximum number of jobs that can be concurrently active from the job queue

- For a work station entry:
  - Work station name or work station type
  - Job description name
  - Maximum number of jobs that can be concurrently active for the entry
  - Name of the display file and record format used to obtain the routing data if the routing data is not specified in the job description
  - Whether a user may sign on to a sign-on screen to enter the subsystem or only transfer in from another subsystem

- For a communications entry:
  - Communications device description name
  - Job description name
  - Default user profile name for an attach that contains no password
  - Maximum number of jobs that can be concurrently active through this communications entry

IBM-Supplied Subsystem Descriptions

The following subsystem descriptions are shipped with the system:

- QCTL–controlling subsystem

- QINTER–interactive subsystem

- QBATCH–batch subsystem

- QPGMR–programmer subsystem

- QSPL–spooling subsystem

QINTER supports all of the interactive jobs processed through the display work stations on the system. Either QCTL or a user-defined subsystem can be the controlling subsystem.

QBATCH supports all batch jobs processed on the system. Any batch subsystem must be started by a Start Subsystem (STRSBS) command and terminated by a Terminate Subsystem (TRMSBS), Terminate CPF (TRMCPF), or Power Down System (PWRDWNSYS) command.

QSPL supports reading jobs and job streams and then writing the output from the jobs.

Note: See *Work Monitor* for additional information about subsystems and subsystem descriptions.

Subsystem Description External Controls

A subsystem description can be controlled externally by CL commands. There are commands to create, display, and delete subsystem descriptions. They are:

- Create Subsystem Description (CRTSBSD)
- Display Subsystem Description (DSPSBSD)
- Delete Subsystem Description (DLTSBSD)

There are also CL commands to change the contents of a subsystem description. Those commands are:

- Change Subsystem Description (CHGSBSD)
- Add Autostart Job Entry (ADDAJE)
- Change Autostart Job Entry (CHGAJE)
- Remove Autostart Job Entry (RMVAJE)
- Add Communications Entry (ADDCMNE)
- Change Communications Entry (CHGCMNE)
- Remove Communications Entry (RMVCMNE)
- Add Job Queue Entry (ADDJOBQE)
- Change Job Queue Entry (CHGJOBQE)
- Remove Job Queue Entry (RMVJOBQE)
- Add Work Station Entry (ADDWSE)
- Change Work Station Entry (CHGWSE)
- Remove Work Station Entry (RMVWSE)
- Add Routing Entry (ADDRTGE)
- Change Routing Entry (CHGRTGE)
- Remove Routing Entry (RMVRTGE)

GENERAL OVERVIEW

Subsystem Description Modules

The subsystem description component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QWDCADA–Add Autostart Job Entry (ADDAJE)[1]: This module processes the Add Autostart Job Entry command.

-->QWDCADC–Add Communications Entry (ADDCMNE)[1]: This module processes the Add Communications Entry command.

-->QWDCADJ–Add Work Station Entry (ADDWSE)[1]: This module processes the Add Work Station Entry command.

-->QWDCADQ–Add Job Queue Entry (ADDJOBQE)[1]: This command processes the Add Job Queue Entry command.

-->QWDCADR–Add Routing Entry (ADDRTGE)[1]: This module processes the Add Routing Entry command.

-->QWDCCHA–Change Autostart Job Entry (CHGAJE)[1]: This module processes the Change Autostart Job Entry command.

-->QWDCCHC–Change Communications Entry (CHGCMNE)[1]: This module processes the Change Communications Entry command.

-->QWDCCHD–Change Subsystem Description (CHGSBSD)[1]: This module processes the Change Subsystem Description command.

-->QWDCCHG–Change Job Description (CHGJOBD)[1]: This module processes the Change Job Description command.

-->QWDCCHJ–Change Work Station Entry (CHGWSE)[1]: This module processes the Change Work Station Entry command.

[1]This module is a CPP (command processing program).

-->QWDCCHQ–Change Job Queue Entry (CHGJOBQE)[1]: This module processes the Change Job Queue Entry command.

-->QWDCCHR–Change Routing Entry (CHGRTGE)[1]: This module processes the Change Routing Entry command.

-->QWDCCRD–Create Subsystem Description (CRTSBSD)[1]: This module processes the Create Subsystem Description command.

-->QWDCCRG–Create Job Description (CRTJOBD)[1]: This module processes the Create Job Description command.

-->QWDCDLJ–Remove Work Entry (RMVAJE, RMVCMNE, RMVWSE, RMVJOBQE)[1]: This module processes the Remove Autostart Job Entry, Remove Communications Entry, Remove Work Station Entry, and Remove Job Queue Entry commands.

-->QWDCDLR–Remove Routing Entry (RMVRTGE)[1]: This module processes the Remove Routing Entry command.

-->QWDCDSD–Display Subsystem Description (DSPSBSD)[1]: This module processes the Display Subsystem Description command.

-->QWDCDSG–Display Job Description (DSPJOBD)[1]: This module processes the Display Job Description command.

QWDGMSG–Send Messages and Signal Exceptions: This module sends messages and signals exceptions for job description CPPs.

QWDJDVF–Check Validity of a Specified Device File: This module checks the validity of the device file and record format specification.

QWDJFNE–Find Job Entry: This module finds the specified subsystem description job entry.

QWDMMSG–Send Messages and Signal Exceptions: This module sends messages and signals exceptions for subsystem description CPPs.

QWDMXTND–Extend the Subsystem Description: This module extends the size of the subsystem description objects that are extendable.

The following modules are invoked by the ?PCKSBSD and ?WDRSLOBJ macros, respectively:

QWDMPACK–Pack Subsystem Description: This module performs the pack function for the subsystem description.

QWDMRSLV–Resolve to External Objects: This module performs the resolve function for the subsystem description modules.

The following modules are invoked by the ?RCRSBSD macro:

QWDMCNVT–Convert Subsystem Description: This module converts the subsystem description internal format to support multiple job queue entries and communications entries.

QWDMRCVR–Recover Subsystem Description: This module performs the recovery procedures for subsystem description objects.

[1]This module is a CPP (command processing program).

Subsystem Description Overview

Figure WD-1 and the following text describe a high-level overview of the subsystem description component.

1 Operations on a subsystem description are command-driven.

2 The command processing program process takes care of removes, creates, displays, and changes. The command processing program makes changes to an inactive subsystem description.

3 The monitor process makes changes to active subsystem descriptions.

4 Messages are sent as appropriate.

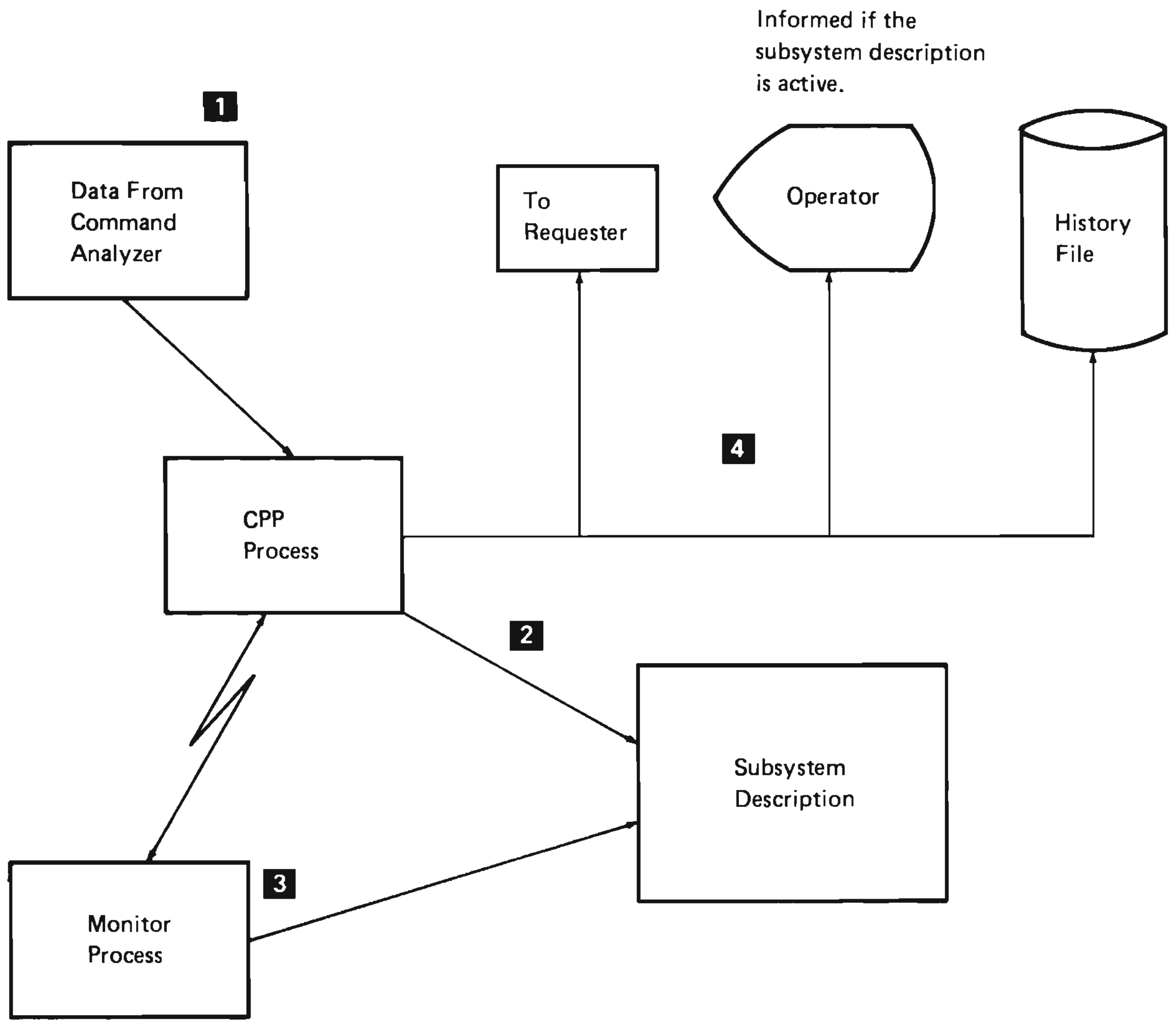


Figure WD-1. High-Level Overview of the Subsystem Description Component

Subsystem Description Internal Structure

Although the user sees the subsystem description as a single object and operates on it as such, the internal structure of a subsystem description is that of a standard composite object, whose pieces are spaces. Figure WD-2 shows the structure of a subsystem description, including the names of the includes that specify the declare structure for each of the objects.

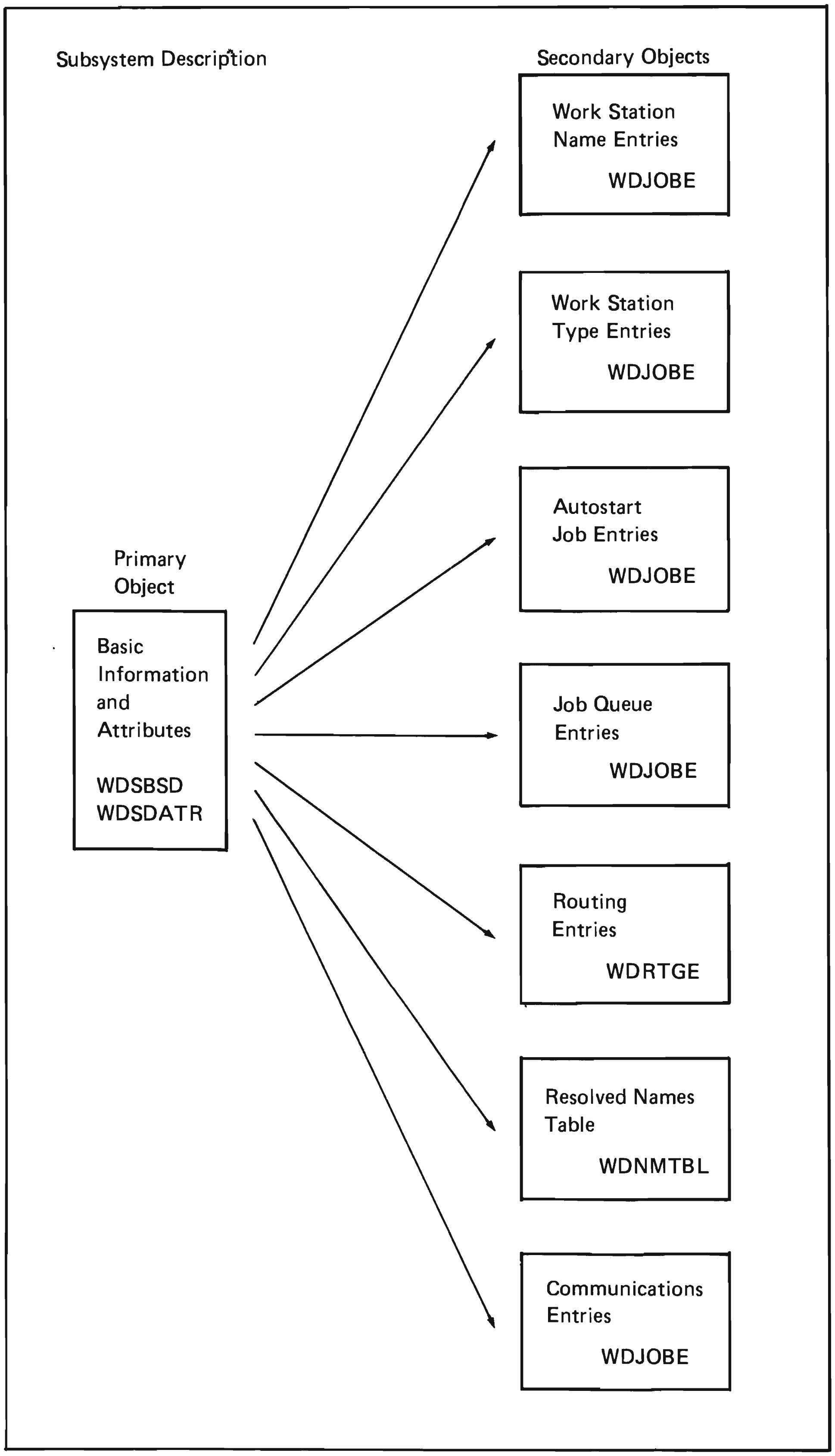


Figure WD-2. Internal Structure of a Subsystem Description

Changing a Subsystem Description

Figures WD-3 and WD-4 show how an inactive and an active subsystem description can be changed.

Inactive Subsystem Description

If a subsystem description is inactive, the change is made by the appropriate command processing program.

Active Subsystem Description

If a subsystem description is active, the change is made by the subsystem monitor.

1 The appropriate command processing program signals the monitor.

2 The monitor process makes the change to the subsystem description.

3 The monitor signals the command processing program when the change to the subsystem description has been made.

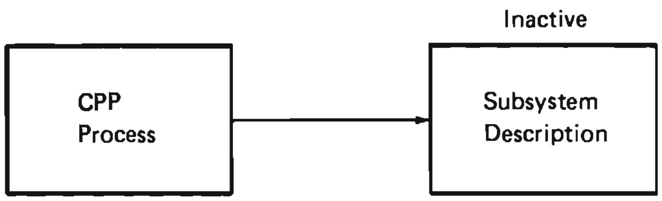


Figure WD-3. Changing an Inactive Subsystem Description

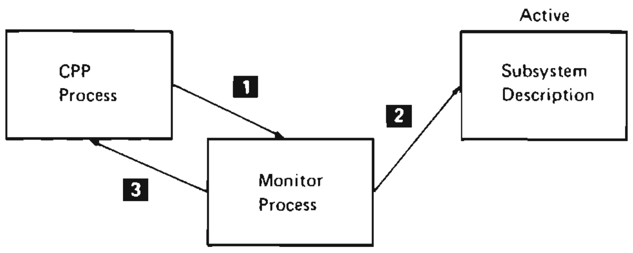


Figure WD-4. Changing an Active Subsystem Description

In both cases of changing a subsystem description (inactive and active subsystems) the code to change the subsystem description is contained in an include segment that is invoked from either the command processing program or the monitor process, as appropriate.

Packing

When a subsystem description work entry or routing entry is flagged as being changed or deleted, a packing-is-required flag is turned on for the affected object. This flag indicates that space can be reclaimed by eliminating the changed or deleted object entry(s).

Reclaiming of this space occurs at the following times:

When a subsystem is started: All areas of the subsystem description are packed, because once the subsystem is active, packing is inhibited.

When there is not enough room left in the subsystem description to accommodate a new entry or a change to an existing entry: Only that area of the subsystem description affected by the change or addition is packed. The packing operation is inhibited when the subsystem description is active. It is inhibited so as not to disturb the present location of entries presently being used by the monitor.

Figure WD-5 shows the objects in a subsystem description that can be packed and those objects that are not subject to packing.

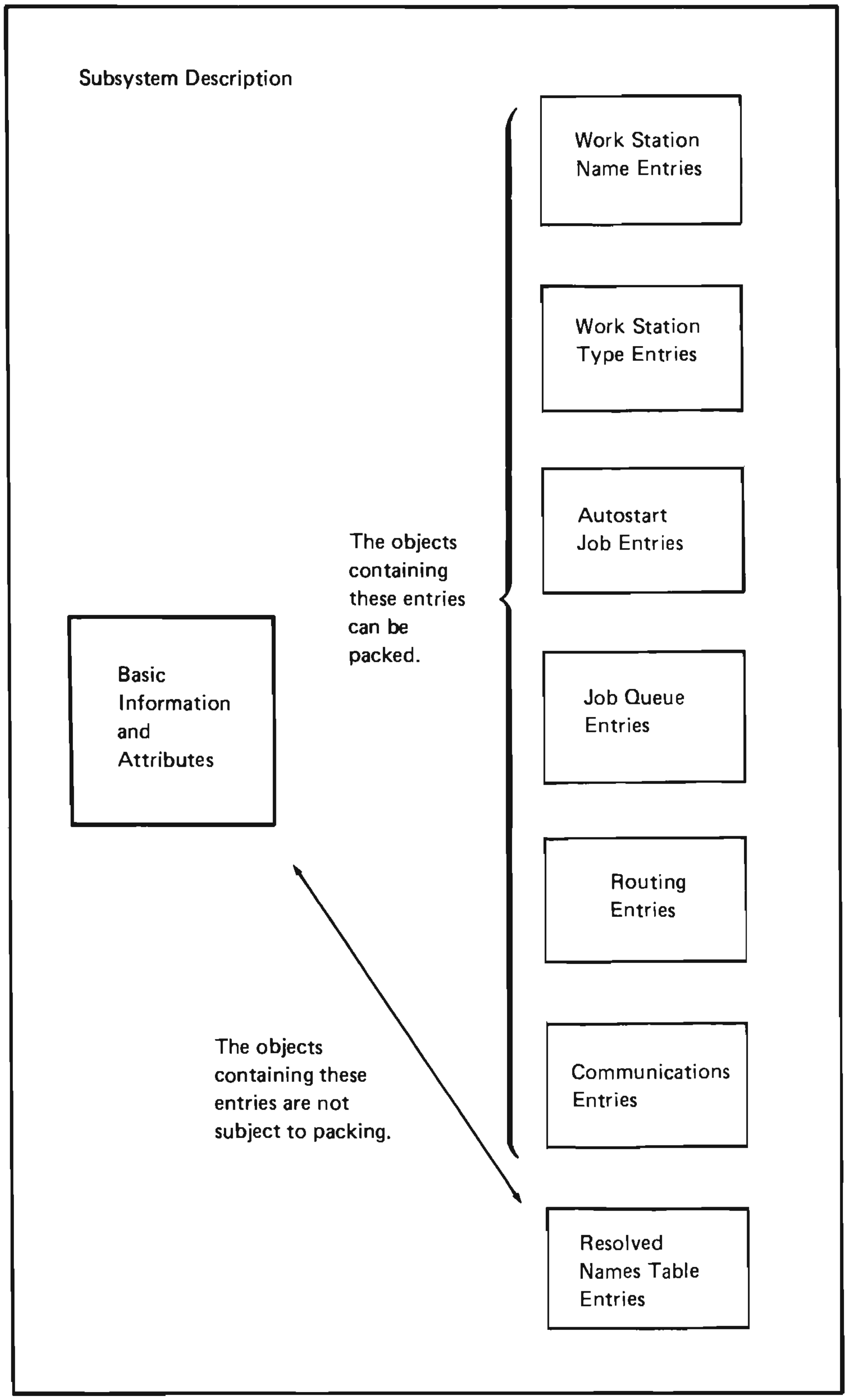

Subsystem Description
Work Station Name Entries
Work Station Type Entries
Autostart Job Entries
Job Queue Entries
Routing Entries
Communications Entries
Resolved Names Table Entries
Basic Information and Attributes
The objects containing these entries can be packed.
The objects containing these entries are not subject to packing.

Figure WD-5. Subsystem Description Objects That Can and Cannot Be Packed

Extending a Subsystem Description

Whenever a subsystem description entry is added or changed, the possibility exists that there will not be enough room left in the subsystem description to accommodate the added or changed data in the affected object.

Should there not be enough space, packing is first attempted. However, the packing-is-required flag must be on and the subsystem description must be inactive. If these two conditions are not met or if insufficient space was freed by packing, extension takes place.

There are rules governing extension. They are:

- Only the actual number of bytes required is requested. (The system will probably return more than is needed.)
- Only that area of the subsystem description that is actually affected is extended.
- All extensions are permanent. (If at a later time the extended object no longer needs the space assigned to it by the extension, that extra space is not returned to the system.)

Figure WD-6 shows the subsystem description objects that can be extended and those objects that can not be extended.

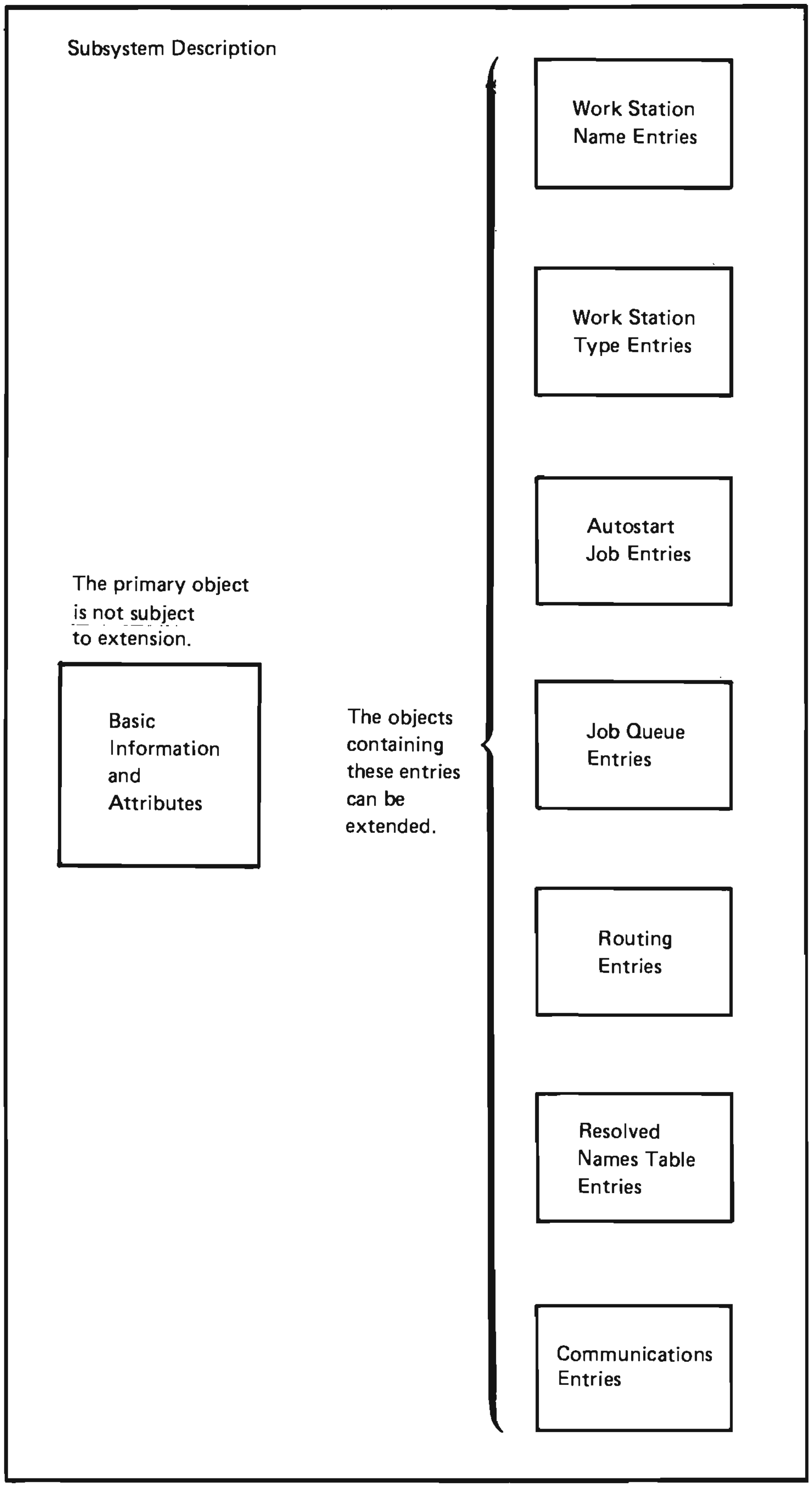

Subsystem Description
The primary object is not subject to extension.
Basic Information and Attributes
The objects containing these entries can be extended.
Work Station Name Entries
Work Station Type Entries
Autostart Job Entries
Job Queue Entries
Routing Entries
Resolved Names Table Entries
Communications Entries

Figure WD-6. Subsystem Description Objects that Can and Cannot Be Extended

Subsystem Description Recovery

The recovery procedures for subsystem descriptions preserve the usability of a subsystem description without requiring that the subsystem description be restored or reinstalled.

All of the data needed to change a subsystem description and to restore it (should the system fail in the middle of a change-in-progress) is contained in a fixed area within the subsystem description primary object.

WWDXRCVY	
Information to change a subsystem description	
Information to restore a subsystem description	
Code (type of change)	WWDXCHG
Status of change	WWDXCHGS
Change-in-progress flag	WWDXCHGP
Recovery-in-progress flag	WWDXRCVP

Contents of these fields vary according to the type of change to be performed.

Recovery Information

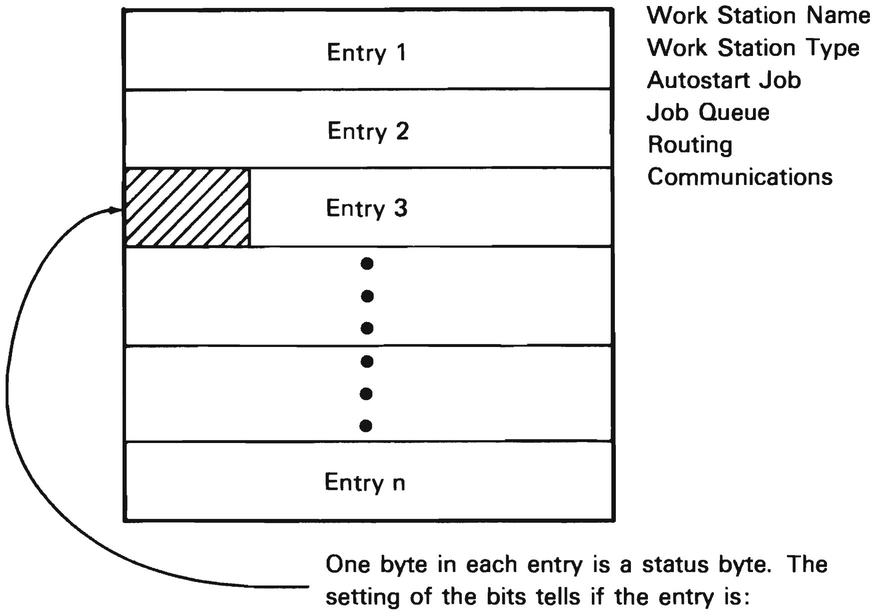
Before changing a subsystem description, all relevant data is placed in a fixed area. When the data is ready to be moved into the affected area of the subsystem description, a change-in-progress flag is turned on. This flag is turned off when the change is complete.

Should a system failure occur while the change-in-progress flag is on, a subsystem recovery module is invoked when the subsystem description is next referenced (after system operation is restored). This module backs out the aborted change (except for removes and attribute changes, which will be carried forward). The recovery-in-progress flag is turned on whenever recovery is in process. The code and status data inform the recovery module what actions are necessary for recovery.

A system failure or any other failure occurring while the recovery-in-progress flag is on results in a damaged object condition for that subsystem description.

Subsystem Description Control Information Entries

The control information specified in many of the CL commands, which affect subsystem descriptions, exist in the form of a series of entries.



Work Station Name
Work Station Type
Autostart Job
Job Queue
Routing
Communications

One byte in each entry is a status byte. The setting of the bits tells if the entry is:

Active–Valid for use
Deleted–Not to be used
Changed–An update to the entry exists further down in the list (this entry is not to be used)
Last–The end of the list

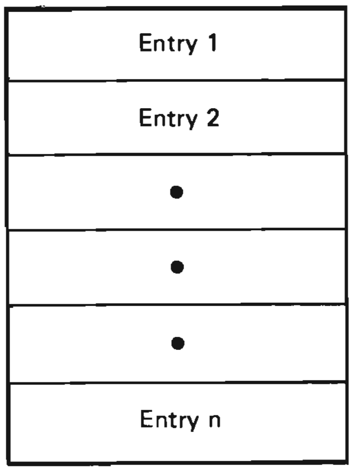
Entries that are deleted or changed are not removed from the list. Rather, their changed or deleted status is flagged. Later, when additional space is needed to add or change an entry, or when a subsystem is started using the subsystem description, the flagged entries are eliminated and the remaining entries are packed. If an entry is changed, its replacement can be found further down the list.

Subsystem Description Entry Structure

Work Entries

The work entries contained in a subsystem description have the following characteristics:

- Order does not matter
- Entries are not frequently referenced



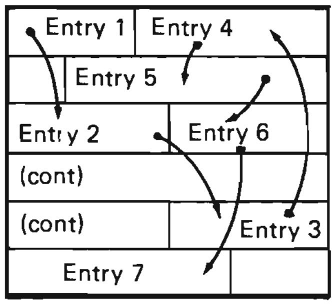
Fixed Length

Note: The length is fixed by the kind of work entry (work station name, autostart, etc.).

Routing Entries

The routing entries in a subsystem description have somewhat different characteristics than work entries. Routing entry characteristics are as follows:

- Order does matter
- Entries more frequently referenced



Routing entries are variable length to reduce the amount of space required. Some information (that which is not immediately required by a monitor) is kept in a separate resolved names table object.

Note: Routing entries are linked in ascending sequence number order. No deleted or changed entries (required to be present within the object because of restrictions imposed by active subsystems) are present on the chain.

Figure WD-7 shows the organization of routing entries.

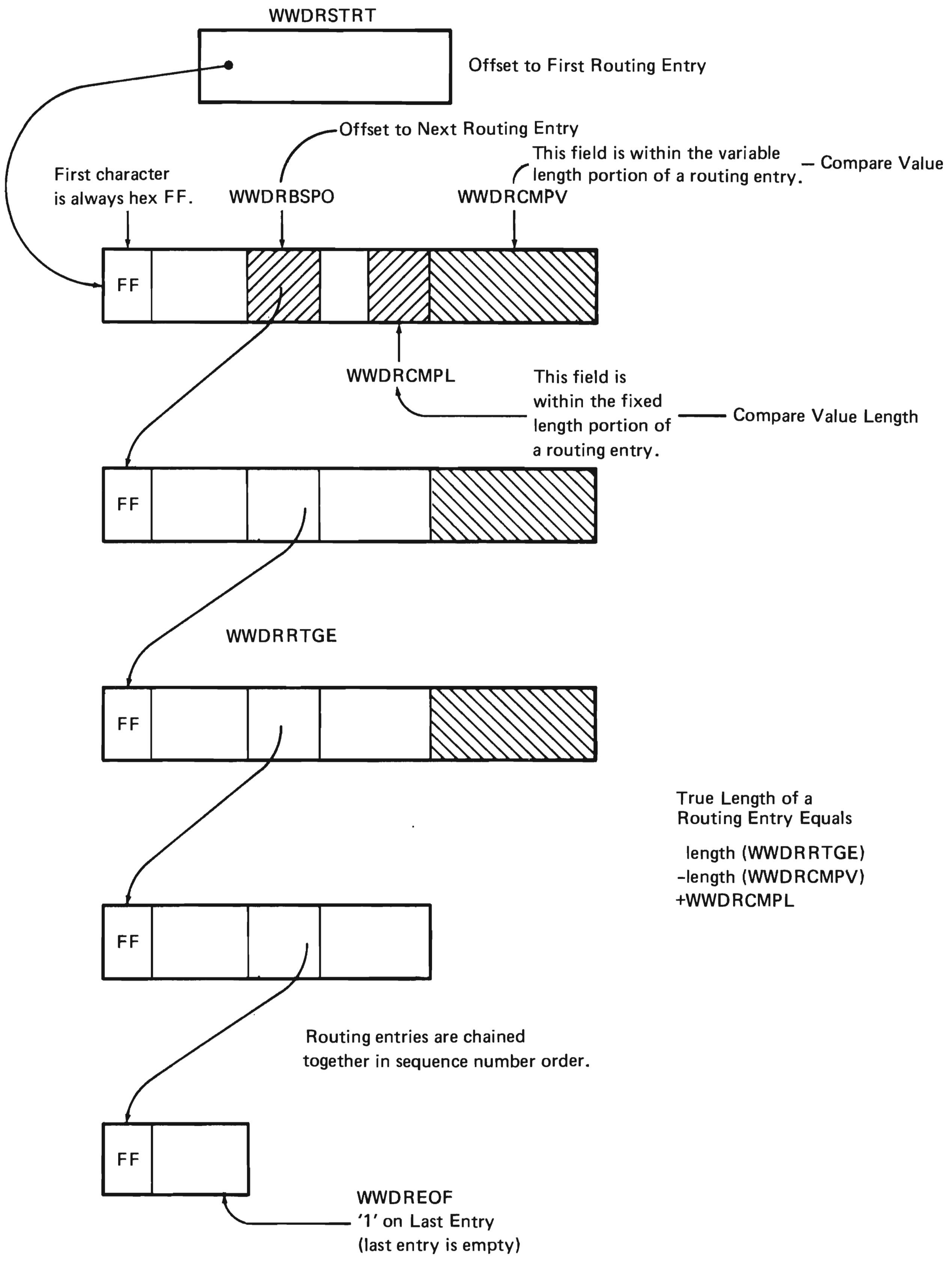
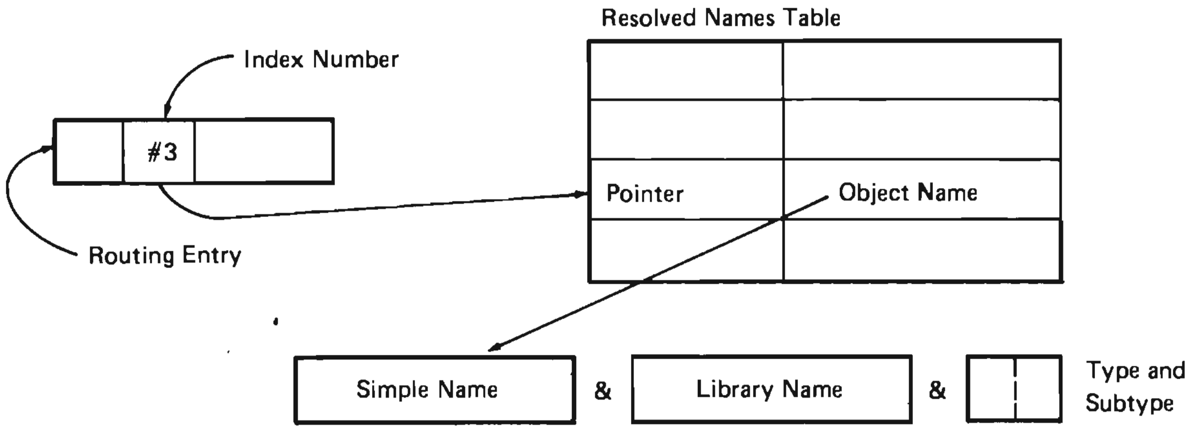

WWDRSTRT
Offset to First Routing Entry
Offset to Next Routing Entry
This field is within the variable length portion of a routing entry.
Compare Value
First character is always hex FF.
WWDRBSPO
WWDRCMPV
FF
WWDRCMPL
This field is within the fixed length portion of a routing entry.
Compare Value Length
FF
WWDRRTGE
FF
True Length of a Routing Entry Equals
length (WWDRRTGE)
-length (WWDRCMPV)
+WWDRCMPL
FF
Routing entries are chained together in sequence number order.
FF
WWDREOF
'1' on Last Entry
(last entry is empty)

Figure WD-7. Organization of Routing Entries

Resolved Names Table

In order to save space in the routing entries secondary object, all information related to external objects is kept in another secondary object called the resolved names table.



Resolved Names Table

When entries are added to a subsystem description, an attempt is made to resolve to the objects named in the command. Routing entries contain index numbers to resolved names table entries. In the resolved names table entry will be found:

- A pointer to the named object.
- The object name. (If the command indicated that an NRL search was to be performed to locate the object, the library name field will be filled in with the information returned from the resolve.)

During subsystem operation, the monitor normally uses only the pointer to the object. However, this pointer could be unusable (perhaps the resolve attempt made when the entry was added was not successful or perhaps the object originally resolved to no longer exists, as happens when programs are recompiled). If the pointer is unusable, the subsystem monitor will attempt to resolve, using the object name information stored in the resolved names table entry. If this fails, a diagnostic message is issued.

Note: Only active routing entries refer to resolved names table entries. If a routing entry is changed or deleted, its name table entries are made available for reuse. Moves and renames of programs and classes specified in routing entries do not make the pointer unusable. The Display Subsystem Description command displays the name that will be used to resolve and the names that were originally specified in the routing entry rather than the name of the program and class as they might exist at present.

Figure WD-8 shows an overview of available entries in the resolved names table.

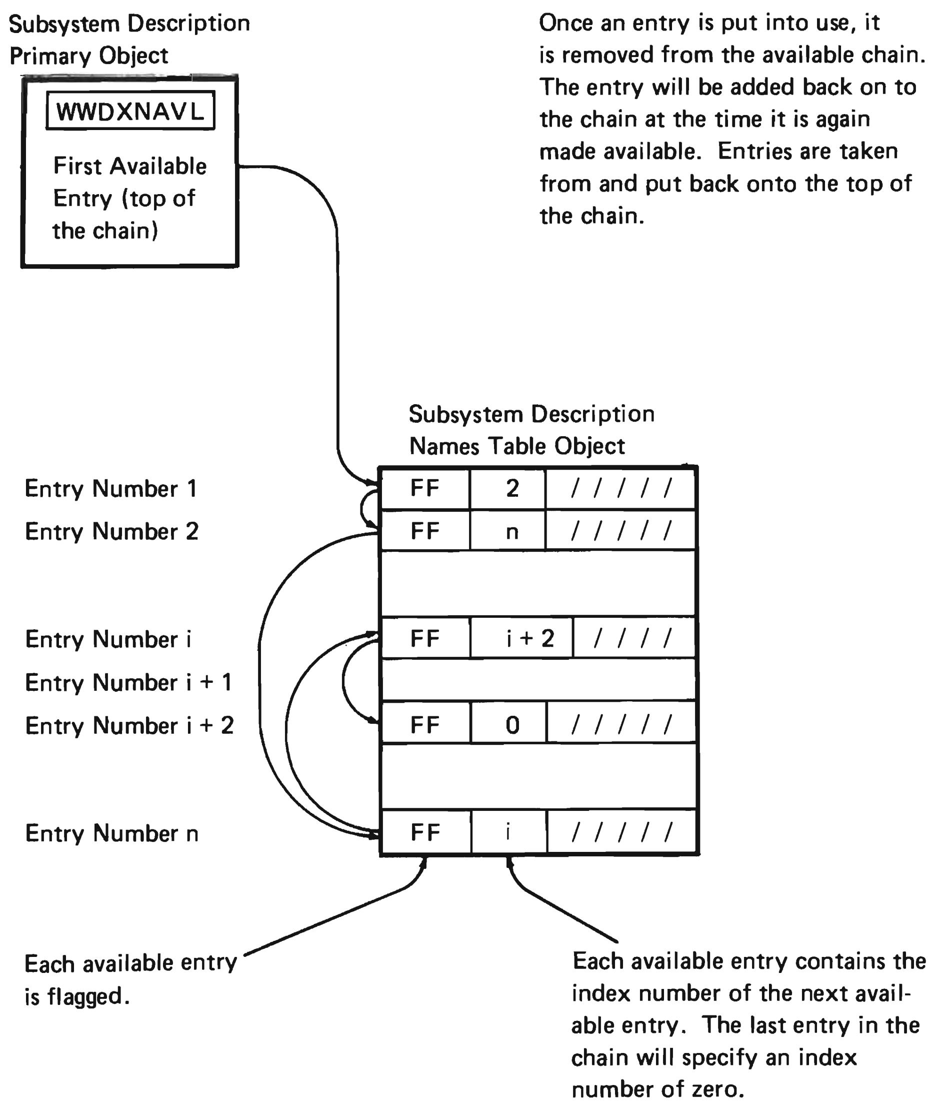

Subsystem Description
Primary Object
WWDXNAVL
First Available Entry (top of the chain)
Once an entry is put into use, it is removed from the available chain. The entry will be added back on to the chain at the time it is again made available. Entries are taken from and put back onto the top of the chain.
Subsystem Description
Names Table Object
Entry Number 1
Entry Number 2
Entry Number i
Entry Number i + 1
Entry Number i + 2
Entry Number n
FF
2
/ / / / /
FF
n
/ / / / /
FF
i + 2
/ / / /
FF
0
/ / / / /
FF
i
/ / / / /
Each available entry is flagged.
Each available entry contains the index number of the next available entry. The last entry in the chain will specify an index number of zero.

Figure WD-8. Resolved Names Table–Available Entries

Flow of Subsystem Description Modules That Change the Subsystem Description

Figure WD-9 and the following text describe the general module flow of subsystem description modules that change the subsystem description.

1 Initialization functions–This includes copying into the CPP (command processing program) the data from the command analyzer and setting up the replacement text and message flags that will be passed to QWDMMSG later.

2 Locate the subsystem description–A separate module performs all resolve functions for this component. Difficulties encountered in locating the subsystem description are noted for later message issuance.

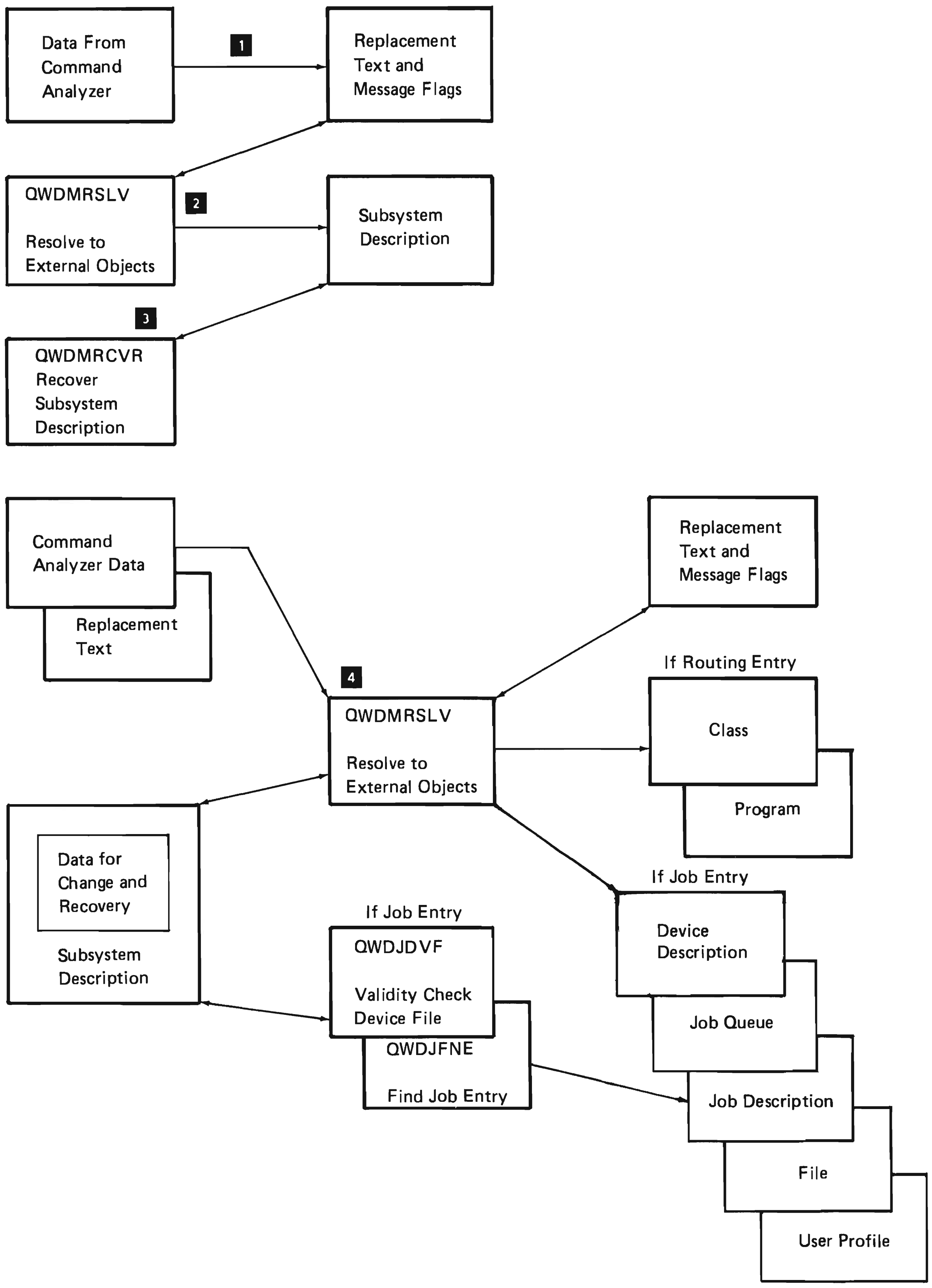
3 Lock the subsystem description–The subsystem description is locked according to protocol. If the subsystem description is not active and it is found that a change-in-progress flag is on in the subsystem description (an indication of possible damage), a separate module is invoked to attempt recovery. Difficulties with the locking protocol are noted for later message issuance.

4 Validation–The data that was passed from the command is analyzed to uncover as many error or warning conditions as is practical. This often involves references to existing data in the subsystem description. External objects specified in the command are located. Data within those objects is used as appropriate to the validation process. An image of the entry to be placed into the subsystem description is built. Both serious and warning conditions are noted for later message issuance.

5 Change the subsystem description–If no serious errors were detected so far, the command processing program proceeds to change the subsystem description. Any remaining data required to change or recover the subsystem description is filled in. If the present size of the subsystem description cannot accommodate the change, then the pack, and if required, the extend modules are invoked to provide the required space. If the subsystem description is not active, the CPP makes the changes to it. Otherwise, the monitor presently using the subsystem description is signaled to make the changes. Difficulties in changing the subsystem description are noted for later message issuance.

6 Unlock the subsystem description.

7 The final step is to invoke the module that will send out the messages indicated by the message flags and, if necessary, to signal an exception to the caller of the CPP. In addition to messages sent to the requester, a record of change activity is sent to the history file. The system operator is also notified if an active subsystem description is changed.



Data From Command Analyzer
1
Replacement Text and Message Flags
QWDMRSLV Resolve to External Objects
2
Subsystem Description
3
QWDMRCVR Recover Subsystem Description
Command Analyzer Data
Replacement Text
Replacement Text and Message Flags
4
QWDMRSLV Resolve to External Objects
If Routing Entry
Class
Program
Data for Change and Recovery
Subsystem Description
If Job Entry
QWDJDVF Validity Check Device File
QWDJFNE Find Job Entry
If Job Entry
Device Description
Job Queue
Job Description
File
User Profile

Figure WD-9 (Part 1 of 2). General Flow of Modules That Change a Subsystem Description

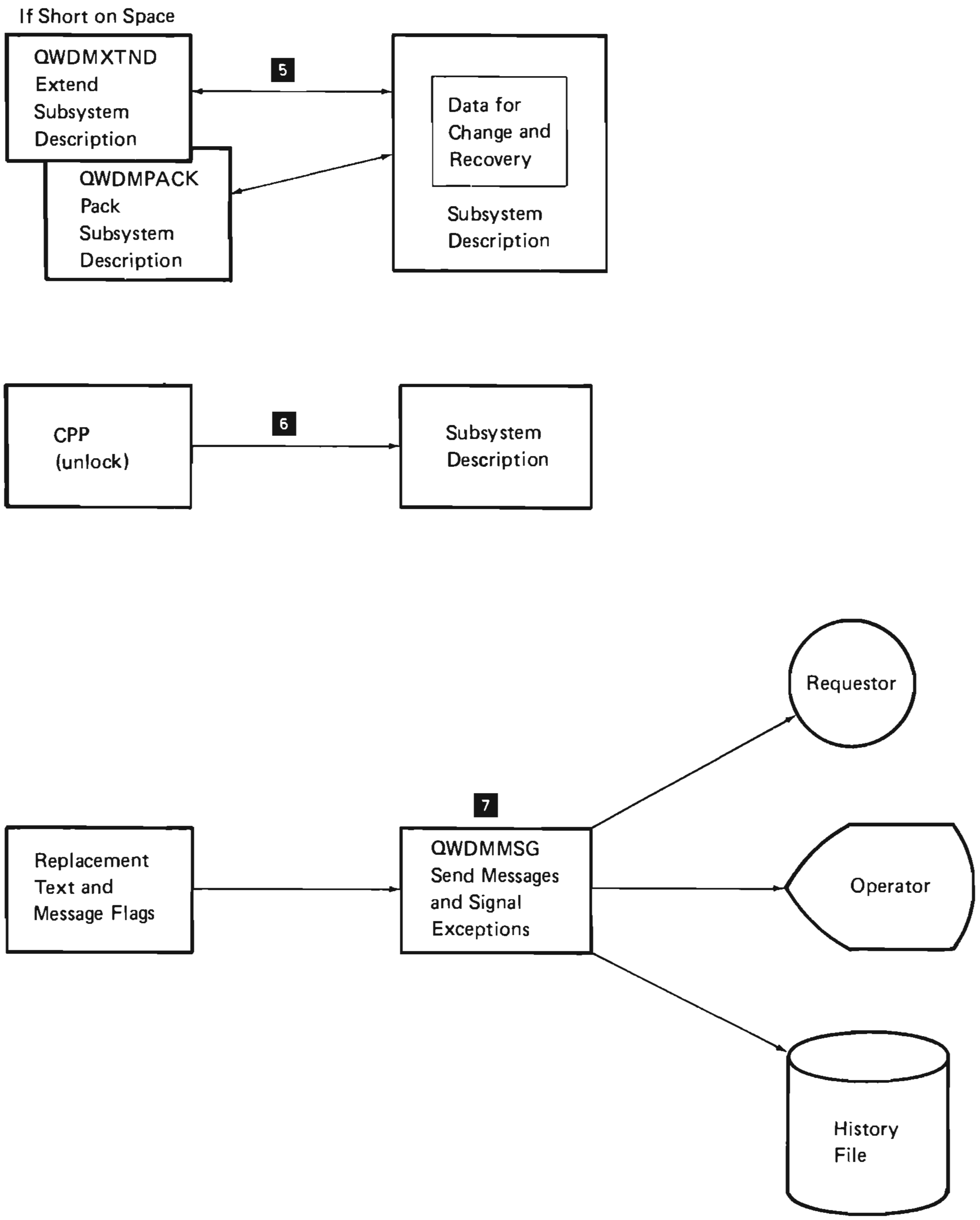

If Short on Space
QWDMXTND Extend Subsystem Description
QWDMPACK Pack Subsystem Description
5
Data for Change and Recovery
Subsystem Description
CPP (unlock)
6
Subsystem Description
Requestor
7
Replacement Text and Message Flags
QWDMMSG Send Messages and Signal Exceptions
Operator
History File

Figure WD-9 (Part 2 of 2). General Flow of Modules That Change a Subsystem Description

File Reference Function

INTRODUCTION

The file reference function component of the CPF (control program facility) processes four CL commands. Each command requests the extraction and display of information concerning various system object attributes. This information can be printed, displayed on a work station, or inserted in a data base file specified by the user. The commands processed by the file reference function component are:

- Display Data Base Relations (DSPDBR)
- Display File Field Description (DSPFFD)
- Display File Description (DSPFD)
- Display Program References (DSPPGMREF)

GENERAL OVERVIEW

File Reference Function Modules

The file reference function component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QWHCLEAN–File Reference Clean-Up: This module is called to cancel any spool files created QWHDSDBR, QWHDSFFD, QWHDSPFD, or QWHDSPPR, when the file reference function is not allowed to complete normally.

-->QWHDSDBR–Display Data Base Relations (DSPDBR)[1]: This module determines the file dependency relationships for the file requested by the user.

QWHOUTFL–Verify/Create Outfile: This module verifies the outfile specified on the Display Data Base Relations, Display File Field Descriptions, Display File Description, and Display Program References commands. If the file or member does not exist, QWHOUTFL will create them.

-->QWHDSFFD–Display File Field Descriptions (DSPFFD)[1]: This module determines the file field level descriptions for each field of the file requested by the user.

QWHOUTFL–Verify/Create Outfile: This module verifies the outfile specified on the Display Data Base Relations, Display File Field Descriptions, Display File Description, and Display Program References commands. If the file or member does not exist, QWHOUTFL will create them.

-->QWHDSPFD–Display File Description (DSPFD)[1]: This module determines the file description for the file requested by the user and processes information that is displayed or printed.

QWHFDOUT–Display Outfile Description: This module processes the file description information to be placed in a data base output file.

QWHOUTFL–Verify/Create Outfile: This module verifies the outfile specified on the Display Data Base Relations, Display File Field Descriptions, Display File Description, and Display Program References commands. If the file or member does not exist, QWHOUTFL will create them.

-->QWHDSPPR–Display Program References (DSPPGMREF)[1]: This module determines the device files, data base files, and CL variables, which are to be device files or data base files, that are referenced by the program requested by the user.

[1]This module is a CPP (command processing program).

-->QWHFDCHK–Display File Description Validity Checker: This module is a validity check program for the Display File Description command. QWHFDCHK verifies that invalid combinations of parameters have not been specified.

QWHOUTFL–Verify/Create Outfile: This module verifies the outfile specified on the Display Data Base Relations, Display File Field Descriptions, Display File Description, and Display Program References commands. If the file or member does not exist, QWHOUTFL will create them.

File Reference Function Overview

Figure WH-1 and the following text show the two invocation paths to the file reference function component. It also shows the CPF components involved in an invocation and those components supporting the file reference function access to data base file descriptions, device file descriptions, and library data. File reference function also uses the 5211/3262/3203 function manager to print the requested information, spooling to display the information, data base and common data management to send the information to a file, and the message handler to handle messages and exceptions.

1 The invocation paths are identical for each of the four commands. Each command, with the desired parameters, can be entered from a console or work station, with or without prompting, or from a batch processing CL job or CL program. Each command is syntax checked by the command analyzer before it is processed by the appropriate file reference function command processing program.

2 QWHFDCHK performs additional checks on the Display File Description command that are not performed by the command analyzer. QWHFDCHK also signals the following errors:

- Multiple TYPEs specified with OUTFILE.
- FILEATR (*MXD) not allowed with TYPE (*ATR) when OUTFILE is specified, or multiple FILEATRs specified with OUTFILE and TYPE (*ATR).
- TYPE does not have a valid FILEATR specification.
- FILEATR does not have a valid TYPE specification.

3 QWHDSDBR extracts, formats, and either prints, displays, or puts in a file, dependency relationships for the displayed data base files. The actual displayed information depends on the file characteristics and the parameter options specified by the user. The three types of dependency lists that can result are:

- A list of all files using the specified format or formats.
- A list of all files sharing a data or access path with the specified file or files.
- A list of all members that share a data or access path with the specified member.

If the information is to be put in a data base file, as specified by the user, and that file does not exist, QWHOUTFL will create the file. QWHDSDBR signals the appropriate exception if any of the following errors are detected:

- The library, file, member, or format was not found.
- The file specified is a device file.
- An error occurred while extracting the file definition, record formats, or member names.
- The output file is not a sequential physical file with record format QWHDRDBR.
- The output file is QADSPDBR.QSYS.
- An output file error occurred during file or member creation, open, put, or close.
- An error occurred while clearing a member of an existing output file.
- The user is not authorized to the file or outfile.
- An error occurred when a display was attempted or when a display was canceled.

4 QWHDSFFD extracts, formats, and either displays, prints, or puts in a file, field level descriptions for each field of the specified data base file or files. The actual displayed information depends on the file characteristics.

If the information is to be put in a data base file, as specified by the user, and that file does not exist, QWHOUTFL will create the file. QWHDSFFD signals the appropriate exception if any of the following errors are detected:

- The library or file was not found.
- An error occurred while extracting the file definition or record formats.
- The output file is not a sequential physical file with record format QWHDRFFD.
- The output file is QADSPFFD.QSYS.
- An output error occurred during file or member creation, open, put, or close.
- An error occurred while clearing a member of an existing output file.
- The user is not authorized to the file or outfile.
- An error occurred when a display was attempted, or when a display was canceled.
- The overflow value or form size width for the print file is too small.

5 QWHDSPFD extracts, formats, and either displays or prints the file descriptors for the data base or device file specified. The actual displayed information depends on the file characteristics and the parameters specified. Any of the following types of file data can be displayed if the type requested is appropriate to the specified file:

- File attributes
- File access paths and key descriptions
- Select and omit specifications
- Collating sequence specifications
- Record format descriptions
- Member descriptions
- Spooling attributes
- Member list

QWHDSPFD signals the appropriate exceptions if any of the following errors are detected:

- The library or file was not found.
- An error occurred while extracting the file attributes, record format names, member names, member attributes or record format attributes.
- An invalid option was entered on the TYPE or FILEATR parameters for the type of file.
- The user is not authorized to the file.
- An error occurred while opening, closing, or writing information to the output file.
- An error occurred when a display was attempted, or when a display was canceled.
- The overflow value or form size width for the print file is too small.

QWHFDOUT performs the same functions as QWHDSPFD, except QWHFDOUT places the output data in a data base file.

QWHFDOUT signals the appropriate exceptions if any of the following errors are detected:

- The library or file was not found.
- An error occurred while extracting file attributes, record format names, member names, member attributes, or record format attributes.
- An invalid option was entered on the TYPE or FILEATR parameters for the type of file.
- The user is not authorized to the file.
- An error occurred while creating, opening, closing, or writing information to the output file.

- The output file is not a sequential physical file with the correct record format.
- The output file is the system supplied file in QSYS.
- An error occurred while clearing a member of an existing output file.

6 QWHDSPPR extracts, formats, and either displays, prints, or puts in a file, a listing of the device files, data base files, and CL variables, which are to be device files or data base files, that are referenced by the displayed program. The actual displayed data depends on the characteristics of the displayed program.

If the information is to be put in a data base file, as specified by the user, and that file does not exist, QWHOUTFL will create the file.

QWHDSPPR signals the appropriate exceptions if any of the following errors are detected:

- The library or program was not found.
- An error occurred while extracting the program references.
- The output file is not a sequential physical file with record format QWHDRPPR.
- The output file is QADSPPGM.QSYS.
- An output file error occurred during file or member creation, open, put, or close.
- An error occurred while clearing a member of an existing output file.
- The user is not authorized to the program or outfile.
- An error occurred when a display was attempted, or when a display was canceled.

7 QWHOUTFL extracts the outfile to verify that it is the correct type of file and has the correct record format. If the output file and member does not exist, QWHOUTFL creates them. QWHOUTFL signals the appropriate exception to the command processing program if any of the following errors are detected:

- The system supplied outfile description is not available.
- The correct record format name is not found in the outfile.
- The outfile is not a physical arrival file.
- The outfile library is not found, authorized, or accessible.
- An error occurred while creating the output file or member.
- The file is a where-used system file.
- An error occurred while clearing a member of an existing output file.

8 QWHCLEAN cancels any spool files created by the file reference command processing program when the command processing program is not allowed to complete execution. QWHCLEAN signals an error occurred when canceling the display.

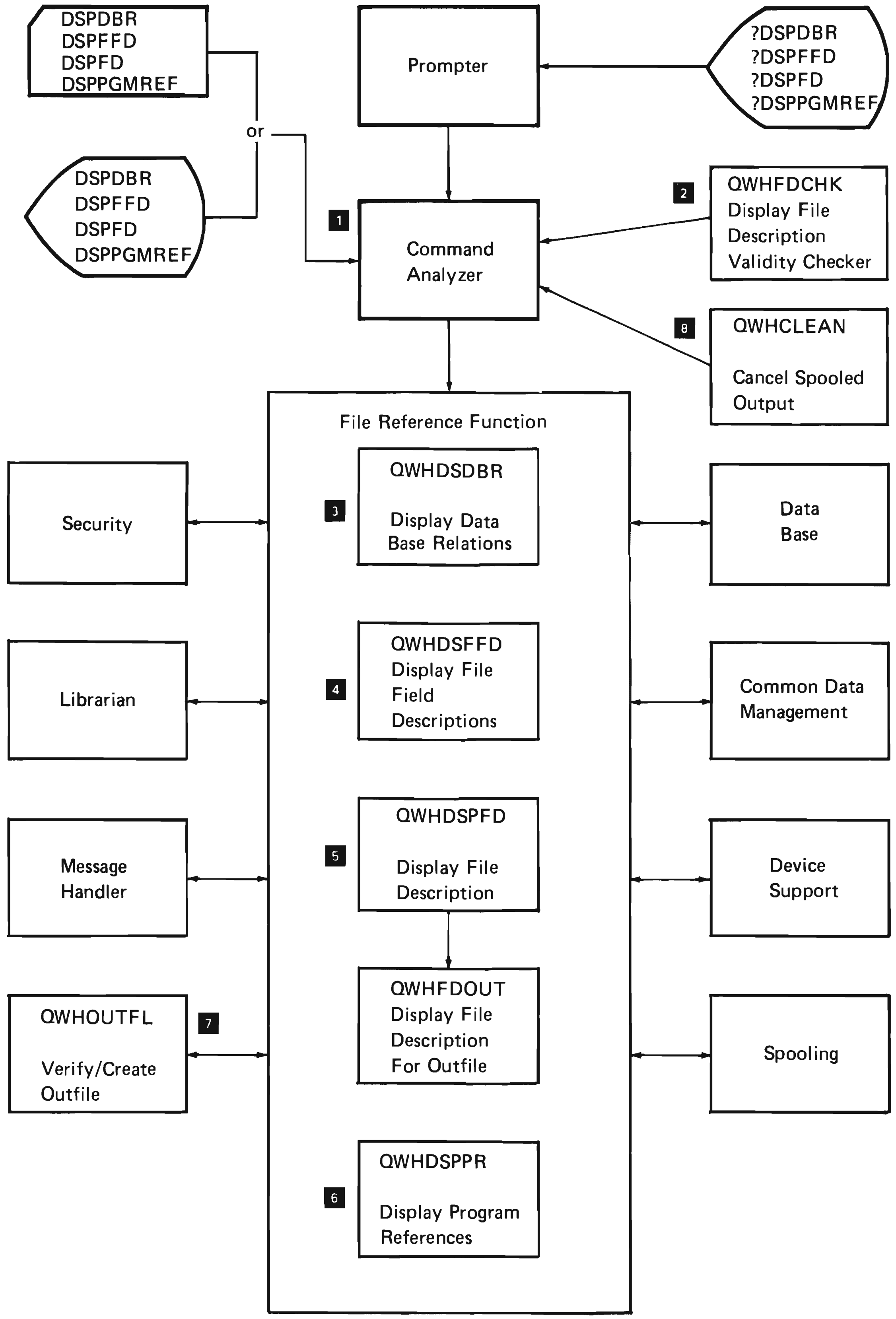

DSPDBR
DSPFFD
DSPFD
DSPPGMREF
or
DSPDBR
DSPFFD
DSPFD
DSPPGMREF
Prompter
?DSPDBR
?DSPFFD
?DSPFD
?DSPPGMREF
1
Command Analyzer
2
QWHFDCHK Display File Description Validity Checker
8
QWHCLEAN Cancel Spooled Output
File Reference Function
3
QWHDSDBR Display Data Base Relations
4
QWHDSFFD Display File Field Descriptions
5
QWHDSPFD Display File Description
QWHFDOUT Display File Description For Outfile
6
QWHDSPPR Display Program References
Security
Librarian
Message Handler
7
QWHOUTFL Verify/Create Outfile
Data Base
Common Data Management
Device Support
Spooling

Figure WH-1. File Reference Function Overview

Work Station Printer Function Manager

INTRODUCTION

The work station printer function manager component of the CPF (control program facility) provides the support for the work station printers on System/38. The following printer functions are supported by the work station printer function manager:

- Open printer file for processing
- Close printer file to processing
- Write data to a printer file

GENERAL OVERVIEW

Work Station Printer Function Manager Modules

The work station printer function manager component consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry into the component. Indentation of a module shows its dependency on a previous module.

-->QWPOPEN–Printer Open: This module prepares an output file for processing by a work station printer. The printer is initialized and the LUD (logical unit description) is modified for the following:
- Forms length
- Forms width
- Lines per inch to print
- Characters per inch to print

If the lines per inch to print is to be changed on the 5256 only, a message is sent to the default message queue to change it.

Open modifies the common data management entry point table when field level support is specified, entering the address for QPNPTFLD in place of QPNPUT. When the spool writer is printing data from the spool output queue, open modifies the common data management entry point table by entering the address of QWPREQIO instead of QPNPUT.

QPNALLOC–Continuation of Open: This module is part of the open process and performs those functions common to open processing for printer files. It is invoked by the work station printer function manager open, the 5211/3262/3203 function manager open, and the spool open to validate the open parameters and establish the function manager work area.

-->QWPCLOSE–Printer Close: This module closes a file to the work station printer. Blocked records are printed if the close is normal; records are purged if the close is not normal. If the close is not temporary, the space objects are destroyed.

-->QWPGRCLS–Printer Graphic Close: This module begins the process of closing a printer graphic file. QWPGREOP is called to process any remaining graphic data. Graphic work space is destroyed and QWPCLOSE is called to complete the closing of the printer graphics file.

-->QPNPUT/QPNPTFLD/QWPGRAPH–Put Records: This module places a single data record into a work station printer output file. Page formatting can be controlled by QPNPUT/QPNPTFLD/QWPGRAPH, by the user program, or from information in a device file depending on whether data records are described in the user program, outside the user program in a device file, or in both places.

When print records are folded or truncated, a message is sent to the job log indicating that occurrence.

QWPORDER–GDF Order Header: This module processes GDF orders when called by the put level printer module QWPGRAPH.

QWPGREOP–Graphic End of Page Handler: This module is called by QWPORDER or QWPGRCLS to move graphic data to a work station printer output file.

-->QWPFEOD–Forced End of Data: This module sends print lines to the work station printer that has been put into the printer ouput file but have not yet been printed.

-->QWPUSEVH–Unsolicited Event Handler: This module handles any unsolicited input from the work station printer.

-->QWPNWEVT–REQIO Complete Event Handler: This module handles the REQIO complete event that is signaled by the machine after a nowait I/O operation.

QWPERROR–Error Handler: This module handles exceptions signaled by the SNA-T3 component, I/O managers, and hardware.

QWPLUDIN–LUD Associated Space Initializer: This module initializes the SNA-T3 dependent areas of the LUD-associated space for the work station printer.

QPNOERRS–Error Handler: This module is called by QWPOPEN when an open parameter error occurs or when an open message must be sent to the default message queue.

QPNPERRS–Error Handler: This module is called by QPNPUT/QPNPTFLD when a put parameter error occurs.

QWPXPRMA–Error Handler/Forms Alignment: This module handles exception conditions and hardware I/O errors as well as forms alignment.

-->QWPREQIO–REQIO Processor: This module is the interface to QT3REQIO when blocked data is to be sent to the work station printer. It completes the control blocks needed by QT3REQIO to issue the REQIO instruction.

This page is intentionally left blank.

Print Operation

Figure WP-1 and the following text describe a work station print operation.

1 A high-level language program or the spooling component, through the QDMCOPEN module of common data management, calls QWPOPEN to prepare a file and, if necessary, initialize the printer for a print operation.

- **A** An argument list is passed that contains a pointer to the UFCB (user file control block) and an index into the ODPCB (open data path control block) to the device being opened.
- **B** A message is sent to the default message queue if a different lines per inch to print is specified.
- **C** If the lines per inch or the forms length are initialized, a request I/O is issued through the SNA-T3 to the I/O manager. Control is returned to the caller.

2 After the file has been opened and the printer initialized, the information to be printed can be sent to the printer. This is accomplished by calling QPNPUT/QPNPTFLD/QWPGRAPH. Page formatting information can be found either in a device file or the user program.

- **A** An argument list is passed that contains pointers to the UFCB, option list, and to control information.
- **B** If an error is detected or the forms need to be positioned at line 1, a message is sent to the default message queue. A message is sent to the job log if print lines are to be truncated or folded.
- **C** Request I/Os are issued through the SNA-T3 to the I/O manager to print the records. Up to an entire page of print lines can be loaded into the data buffer before a print operation is performed.

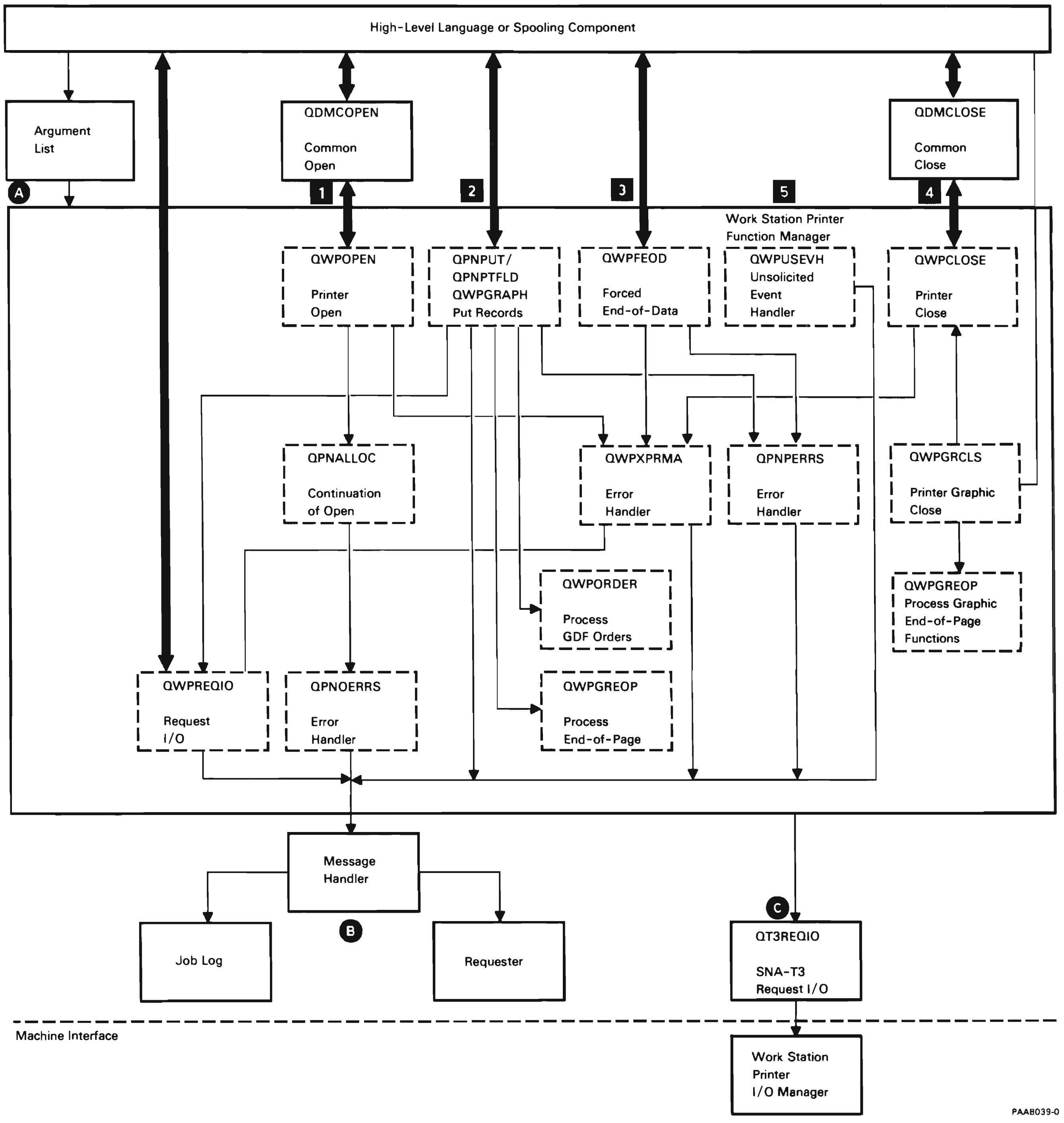
3 QWPFEOD is called to perform a print operation on print lines that have been blocked in the data buffer but not yet printed.

- **A** An argument list is passed that contains a pointer to the UFCB.
- **B** If an error is detected or the forms need to be aligned, a message is sent to the default message queue.

4 After all print records have been passed to the work station printer function manager, QWPCLOSE, through the QDMCLOSE module of common data management, is called to close the file to further processing.

- **A** An argument list is passed that contains a pointer to the ODPCB, the type of close to perform, and an index to the device being closed.
- **B** If an error is detected or the forms need to be aligned, a message is sent the default message queue.
- **C** A printer operation is performed to print those lines that have been blocked in the data buffer but not yet printed.

5 When intervention required is detected by QWPUSEVH, a message is sent to the default message queue. When the appropriate action has been performed and the ready switch on the printer has been pressed, a printer available message is sent to the default message queue and printing resumes.



High-Level Language or Spooling Component
Argument List
QDMCOPEN Common Open
QDMCLOSE Common Close
A
1
2
3
5
4
Work Station Printer Function Manager
QWPOPEN Printer Open
QPNPUT/ QPNPTFLD QWPGRAPH Put Records
QWPFEOD Forced End-of-Data
QWPUSEVH Unsolicited Event Handler
QWPCLOSE Printer Close
QPNALLOC Continuation of Open
QWPXPRMA Error Handler
QPNPERRS Error Handler
QWPGRCLS Printer Graphic Close
QWPORDER Process GDF Orders
QWPGREOP Process Graphic End-of-Page Functions
QWPREQIO Request I/O
QPNOERRS Error Handler
QWPGREOP Process End-of-Page
Message Handler
B
Job Log
Requester
C
QT3REQIO SNA-T3 Request I/O
Machine Interface
Work Station Printer I/O Manager
PAAB039-0

Figure WP-1. Work Station Print Operation

Work Station Printer Function Manager Internal Interfaces

Figure WP-2 and the following text describe the work station printer function manager internal interfaces. The internal interfaces are defined as those events that invoke a work station printer function manager module.

1 The REQIO complete event is signaled whenever an REQIO instruction is completed (direct I/O only).

2 QWPNWEVT dequeues the completed request block from the machine response queue and calls QT3REQIO to process the request block. If the request is a put nowait, a put nowait complete event is signaled to the SNA-T3 component.

3 The unsolicited input event is signaled by the machine whenever it receives data and does not have a request block from SNA-T3 to put the data in.

4 QWPUSEVH handles unsolicited data from the work station printer. It calls SNA-T3 to retrieve the unsolicited input from the machine.

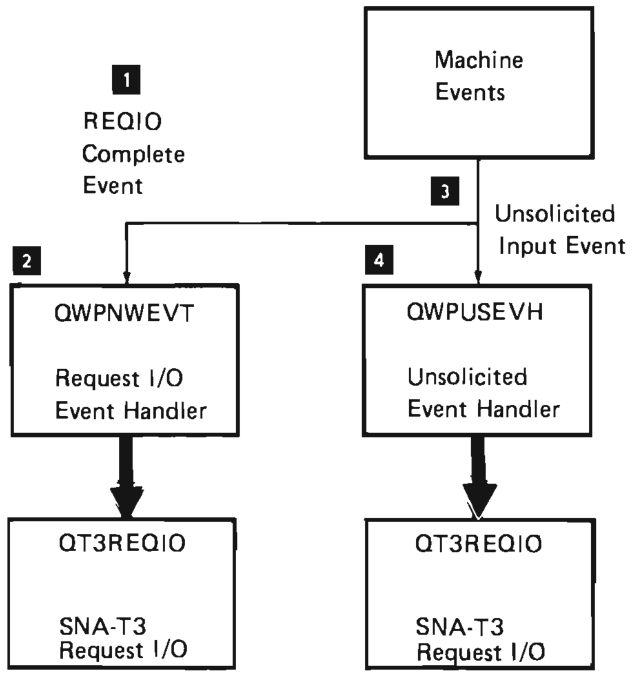


Figure WP-2. Work Station Printer Function Manager Internal Interfaces

5251 Display Function Manager

INTRODUCTION

The 5251 display function manager component of the CPF (control program facility) provides the interface between the display user and display operations. Both sizes of display screens are handled by the 5251 display function manager.

The 5251 display I/O operations are performed through the SNA-T3 component. SNA-T3 actually issues the REQIO instructions and interfaces with the 5251 I/O manager, which is below the machine interface.

The method by which the display is attached to the system (native or remote) is transparent to the function manager. The function manager is responsible for issuing the SNA-T3 requests in the proper sequence, but SNA protocol is enforced by the SNA-T3 component.

Because a process can use multiple displays concurrently, modules of the 5251 display function manager can appear many times in the invocation stack. Each invocation of a function manager module deals with a single display. A user request always identifies the display and a file. Therefore, with those two items being identified, the function manager can perform its functions to the correct device without being aware of the other displays on the system.

GENERAL OVERVIEW

5251 Function Manager Modules

The 5251 function manager consists of the following modules:

Note: An arrow (-->) identifies a module as being an entry module into the component. Indentation of a module shows its dependency on a previous module.

-->QWSCLOSE–Display Close: This module closes a display file to further processing.

   QWSERROR–Display Exception Handling Routine: This module handles exceptions signaled by the SNA-T3 component.

-->QWSTSTR–Test Request: This module handles the Test Request key.

   QWSLUDIN–LUD Initialization: This module initializes the 5251 display and the SNA-T3 dependent areas of the LUD (logical unit description) associated space. This module is invoked only at vary on and LUD creation time.

-->QWSMEEH–Display Nowait Event Handler: This module handles the REQIO Complete event that is signaled by the machine after a nowait I/O operation.

-->QWSMSG–Display Message Function: This module turns on or off the display Message Waiting light.

-->QWSOPEN–Display Open: This module opens a display file to processing.

-->QWSPUT–Put to Display: This module performs the write-to display functions. These functions include writing new data, erasing data on the screen, and unlocking the keyboard.

-->QWSGET–Display Get Function: This module performs the read-from display functions. These functions include such operator requested operations as field validation, printing the display, and second level text.

-->QWSPTMSG–Special Put Message: This module is used by CPF work management to display an operator message independent of the device file open.

-->QWSRST–Restore File: This module restores a suspended file on a display device.

-->QWSSPEND–Suspend File: This module suspends a file on a display device.

   QSFPUT–Put to Subfile: This module puts and updates subfile records and subfile message records.

QSFCRT–Create Subfile: This module creates the subfile and sets any necessary controls for the subfile.

QSFGET–Get From Subfile: This module retrieves records from a subfile.

QSFHSFL–Help Key Support: This module performs the Help key support for a subfile message.

QWSRTSFL–Roll/Truncate/Fold Subfile Support: This module performs the roll and fold or truncate functions.

QWSSFLCT–Display Subfile Control Records Function: This module performs subfile control functions, such as display subfile, display subfile control record, clear subfile, delete subfile, and so forth.

-->QWSUIEH–Unsolicited Input Event Handler: This module handles unsolicited input from the 5251 display.

QSFMQDSP–Message Queue Display: This module handles subfile messages on a program message queue.

-->QWSDSMSG–Display Status Message: This module displays a status message to the display operator.

-->QWSCRTRB–Create Request Block: This module creates the request blocks that are used by the 5251 display function manager.

-->QWSLUDRS–LUD Reset: This module resets portions of the 5251 display dependent area of the LUD-associated space. This module is invoked only at LUD vary off time.

5251 Function Manager External Interfaces

Figure WS-1 and the following text describe the external interfaces to the 5251 function manager. An external interface is defined as a call from the using program to a module within the function manager.

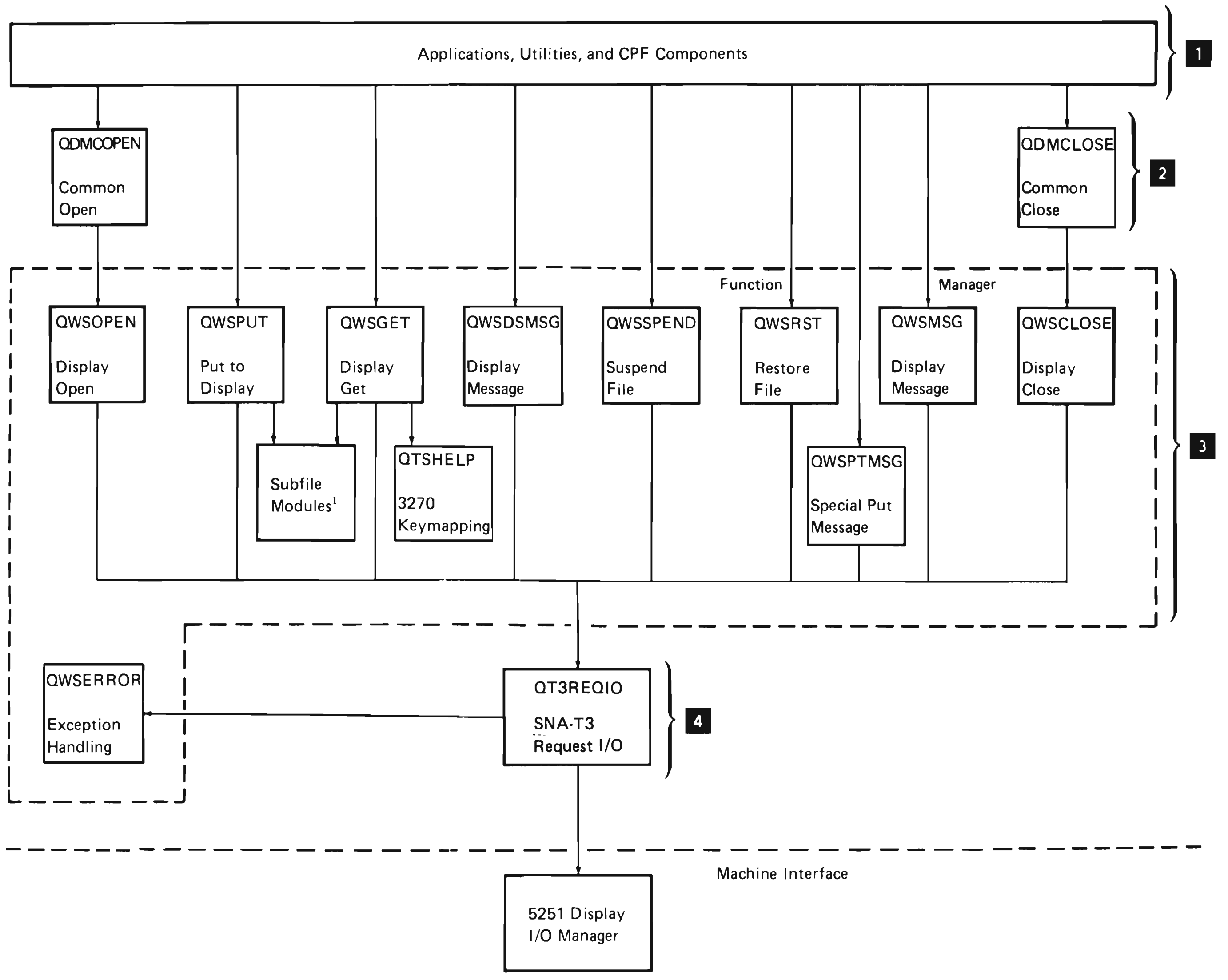
1 Function Manager Users: The function manager does not distinguish one external user from another. A call from an external user identifies a requested function. The function manager performs that function independent of the caller.

2 Upward Interface: All execution time requests and controls from the using program are provided through this interface. The program provides information to the modules through parameters on the call, the UFCB (user file control block), the option list, and the control list. Information is returned in several ways: through the open data path control block (feedback areas), the UFCB (buffer pointers), events, exceptions, and the user input buffer.

3 Function Manager Modules: Each call to a function manager module specifies a single 5251 display. Each invocation of the function manager deals with a single display. Each display is handled as though it was the only display in use.

4 Downward Interface: The downward interface is strictly between the function manager and the SNA-T3 component. It is a controlled interface that is also used by the 5256 function manager component.

- The 5251 function manager always has the SNA-T3 handle the sending of responses to the 5251 requests.
- The function manager only deals with complete chains of data, both sending and receiving.
- Exceptions signaled by SNA-T3 are handled by an external exception handler, QWSERROR.



[1]See Figures WS-9 and WS-10 for detail.

Figure WS-1. 5251 Function Manager External Interfaces

5251 Function Manager Internal Interfaces

The internal interfaces are defined as those events that invoke the 5251 function manager modules.

Figure WS-2 and the following text describe the function manager internal interfaces.

1 The REQIO complete event is signaled whenever put or a get nowait instruction is completed.

2 QWSMEEH dequeues the completion message (feedback record) from the machine interface response queue and calls QT3REQIO to process the T-3 request block. QWSMEEH then determines if the request type was a put or a get. If the request was a put nowait, a put nowait complete event is signaled to the user. If it is a get nowait request, QWSMEEH calls QWSGET to process the input data. QWSGET can do several things: detect an operator error, send a message, and reissue the get nowait request, or they can handle an operator request-help or print. If the get routines process errorless user data, QWSMEEH signals the data available event to the user. If the get routines reissued the get nowait, the data available event is not signaled.

3 The unsolicited input event is signaled by the machine when it receives data and does not have a request block from SNA-T3 to put the data in. This could be from the Help key or the Sys Req/Attn key, which is unsolicited data. Or, it could be that the SIOM received more data than there was room allowed for.

4 QWSUIEH is called by work monitor as a result of the machine unsolicited data event from the 5251 Display as well as from the 5256 printer. It calls SNA-T3 to retrieve the unsolicited input from the machine.

5 QWSTSTR is called by work monitor as a result of the display operator pressing the Test Req key and the display being in the signed-on state. QWSTSTR calls SNA-T3 to retrieve the unsolicited input. QWSTSTR displays a *command key not valid* message to allow the operator to continue.

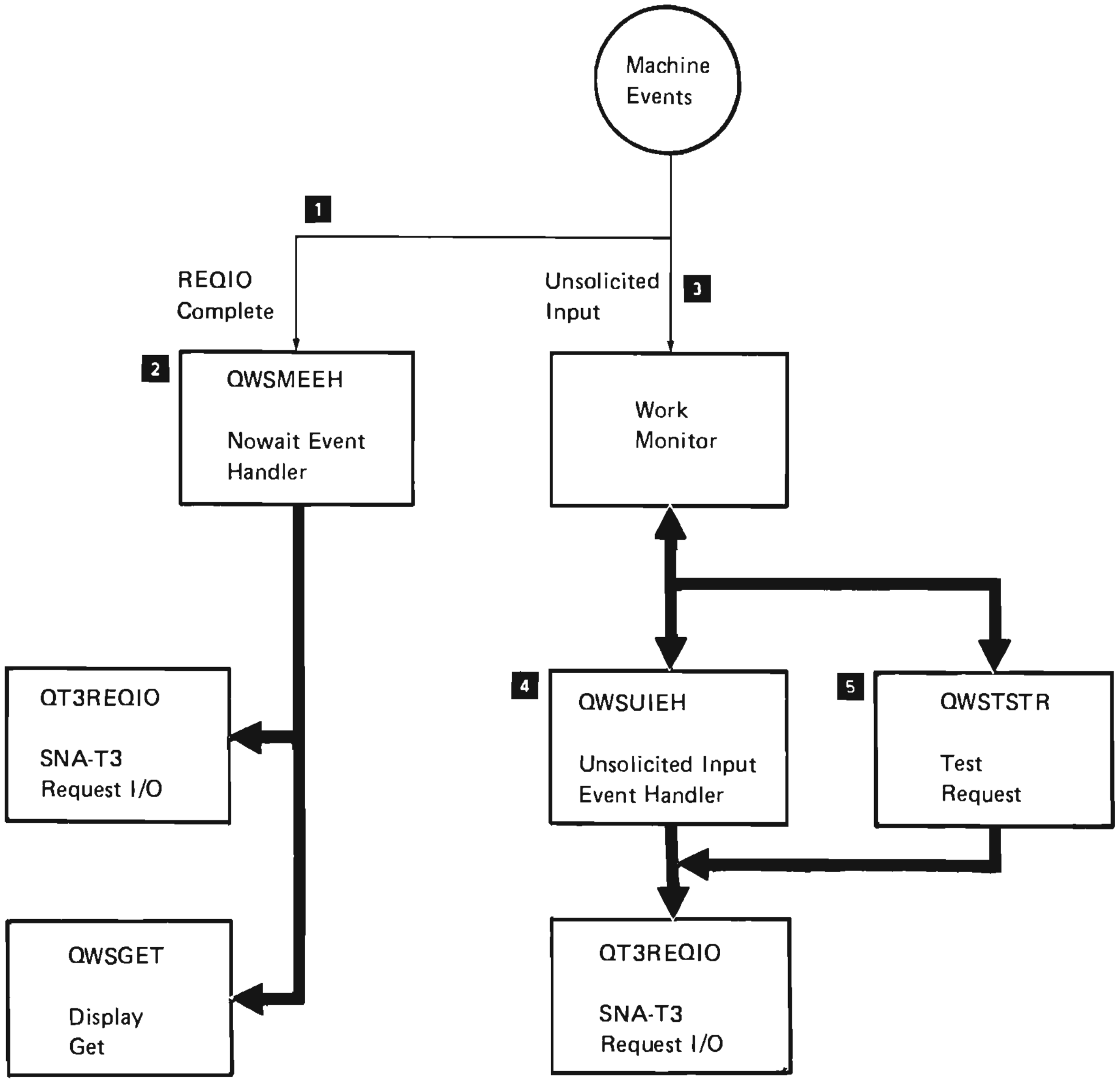


Figure WS-2. 5251 Function Manager Internal Interfaces

Put Operation

Figure WS-3 and the following text describe a put operation.

1 For a field-level put request, the user buffer contains the option indicators followed by all hidden and output fields as described in the device file. Field data is processed one field at a time and is put in the source/sink data area of the SNA-T3 request block, along with the necessary controls to display the data. Hidden fields are saved in the function manager work area.

A nonfield-level put user buffer contains a character string that is treated by the function manager as a single field of data to be sent to the display. The character string is moved from the user buffer to the source/sink data area of the SNA-T3 request block without being looked at or altered. All necessary controls to display the data are supplied by the function manager.

The user buffer for a user-defined data stream request contains the complete 5251 display-dependent data stream. As in a nonfield-level put, the data is moved into the source/sink data area just as it was passed to the function manager. With a user-defined data stream request, the function manager does *not* add display controls to the data.

2 Message File: The user can optionally specify a message from the message file as output data. QWSPUT retrieves the message and sends it to the display user.

3 Device File: Default data and constants from the device file can be sent to the display. The data is taken from the device file, processed and placed in the source/sink data area of the SNA-T3 request block.

4 Source/Sink Data: The source/sink data, which is part of the SNA-T3 request block, is where the function manager builds the 5251-dependent data stream that is to be transmitted. The data stream contains all of the controls needed to display the user record as described in **1**.

5 Function Manager Work Area: The function manager work area is a part of the open data path control block. As the function manager builds the output data stream in the source/sink data area of the SNA-T3 request block, it also builds a user buffer image in the function manager work area. This work area contains a buffer image for each active record on the display screen.

6 Job Log: The device file record as seen in the user buffer can optionally be sent to the job log as well as to the display device.

7 I/O Feedback Area: This area is updated at various times while the function manager is performing the put operation.

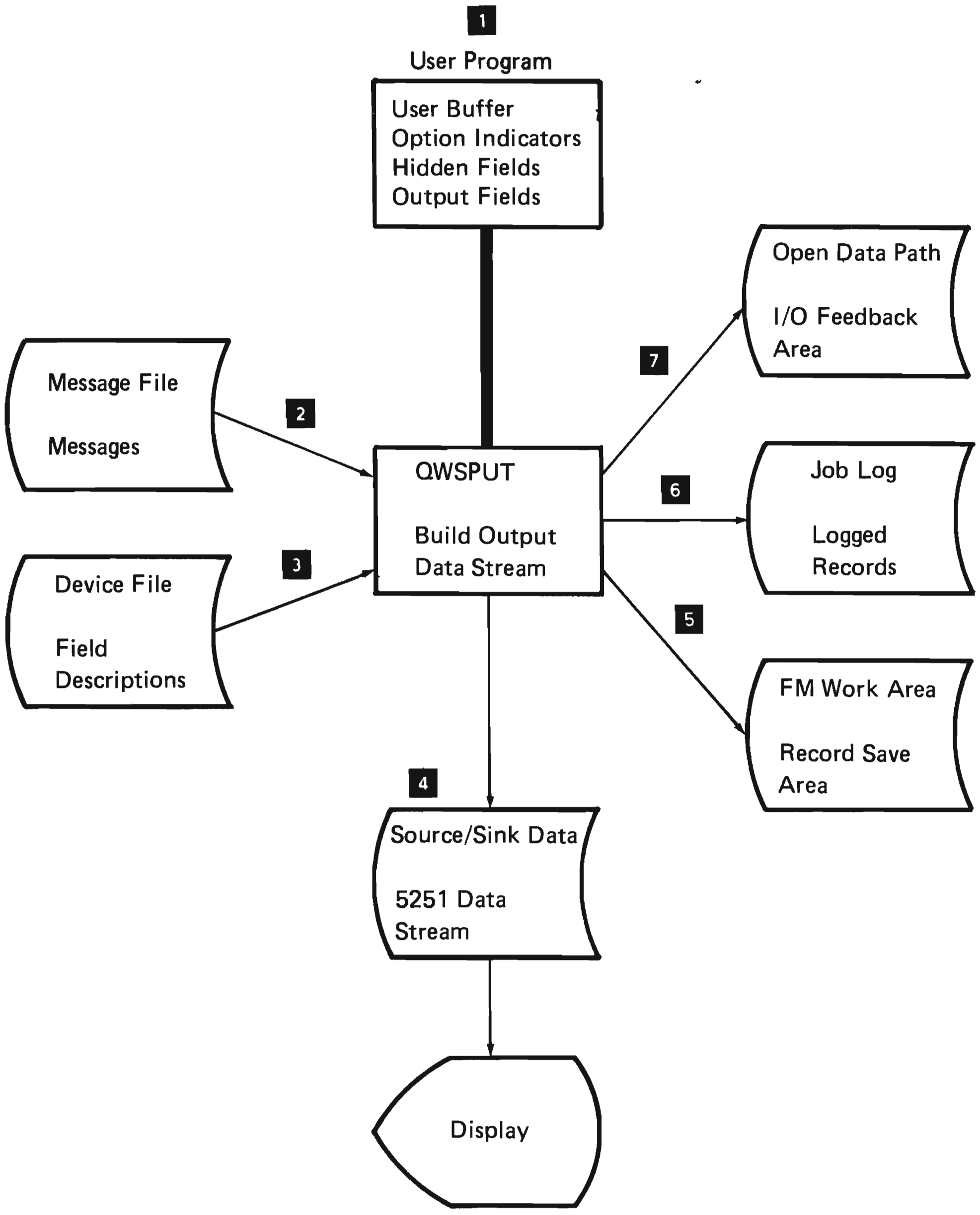


Figure WS-3. Put Operation

Put to Subfile Record (Data Flow)

Only field-level files provide subfile functions. The subfile functions do not support nonfield-level files. User data is treated the same as if the request were to a nonsubfile record by the put routines. QWSPUT determines that the record is a subfile record and transfers control to QSFPUT.

Figure WS-4 and the following text describe a put to a subfile record operation.

1 For a subfile record put request, the user buffer (or a separate indicator area) contains the option indicators followed by all hidden and output fields as described in the device file.

2 The field descriptions for the subfile record are received from the device file and placed in the subfile.

3 QSFPUT takes the data and its attributes, one field at a time, and saves them in the subfile. All of the information needed to display the record as part of the subfile is maintained in the subfile, except for constants that are kept in the device file. The record remains in the subfile until the using program requests that it be displayed.

4 QSFPUT calls QSFCRT during the first subfile record put operation to create the subfile space that will receive all the data from the user buffer or the program message queue if the subfile is a message subfile.

5 The user can optionally specify a message key and a program message queue as a source of data for message subfiles. The subfile modules receive a copy of the messages specified from the message queue and place these into the subfile.

6 QSFPUT calls QSFMQDSP when the put operation to the subfile record specifies that the data to be placed in the subfile should come from a program message queue instead of the user buffer.

7 The subfile is where all subfile information from the user buffer is stored. The subfile record remains in the subfile until the using program requests that it be displayed.

8 The record, as seen in the user buffer, can optionally be sent to the job log as well as to the subfile. This is done independent of the subfile functions.

9 The I/O feedback area is updated at various times while the function manager is performing the put operation. Additional information, such as the relative record operated on, is put in the I/O feedback area for subfile functions.

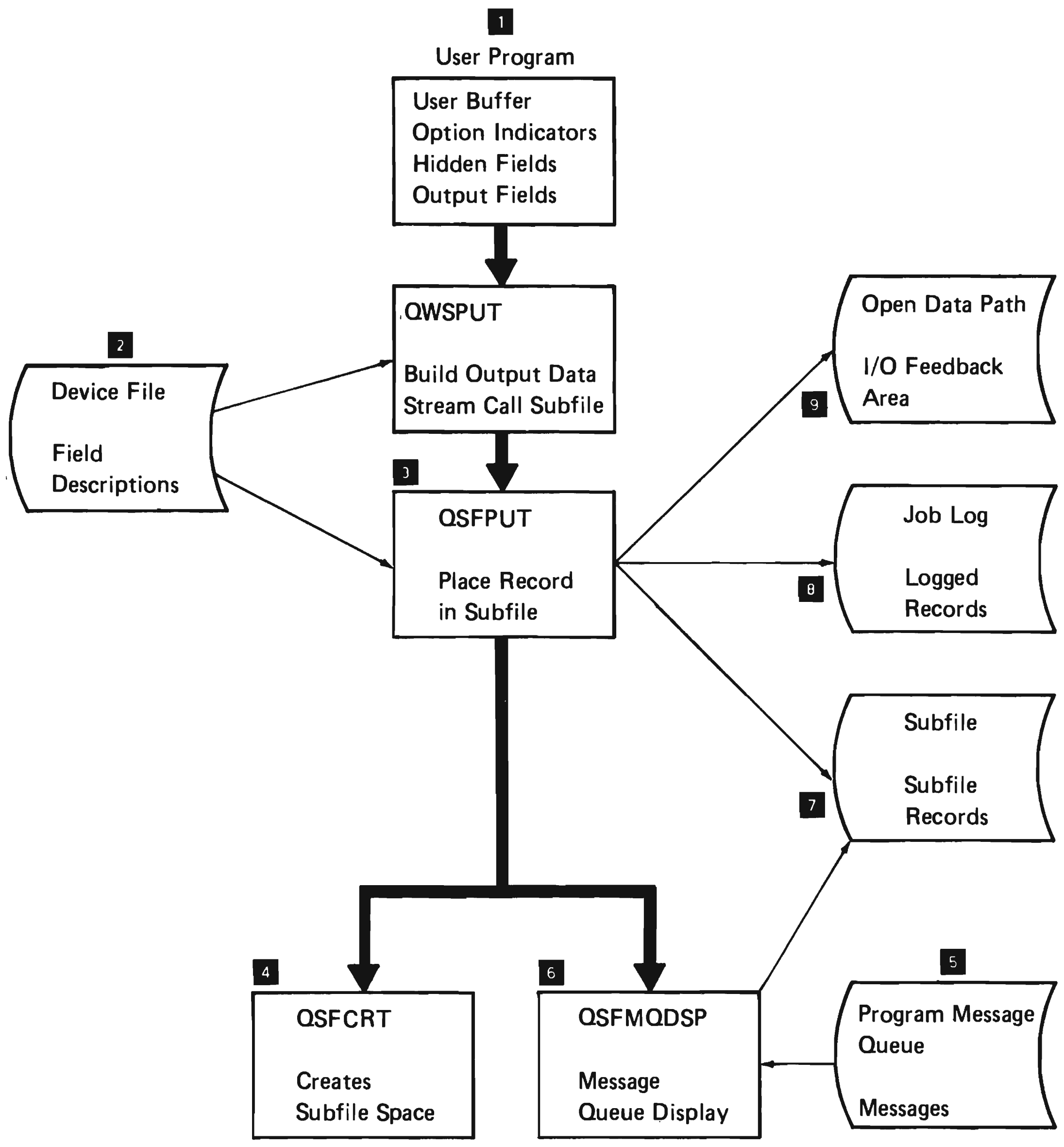


Figure WS-4. Put to Subfile Record (Data Flow)

Put to Subfile Control Record (Data Flow)

Only field-level files provide subfile functions. The subfile functions do not support nonfield-level files. User data is treated the same as if the request were to a nonsubfile record by the put routines. QWSPUT calls QWSSFLCT if a put is done to the subfile control record and a subfile function is requested (such as initialize, delete, and so forth). Operations to the subfile control record only cause I/O to the console if the display subfile or display subfile control record function is indicated.

Figure WS-5 and the following text describe a put to a subfile control record operation.

1 For a subfile control record put request, the user buffer contains the option indicators followed by all hidden and output fields as described in the device file.

2 The function manager processes a put to the subfile control record in two passes. First, the function manager processes a put to the subfile control record as if it were a nonsubfile record. This includes preparing any fields contained in the subfile control record for display by building a data stream, if the display subfile control record keyword is specified. Second, the function manager checks if the display subfile keyword is specified, and if so calls QWSSFLCT to add the subfile records to be displayed to the data stream.

3 If the subfile initialize keyword is specified for the subfile control record, and the subfile space has not been created, QWSPUT calls QSFCRT to create the subfile space for the subfile records. QCOPUT then calls QWSSFLCT to initialize the subfile records from the device file description and displayed by the subfile modules.

4 The constants are used from the device file to build subfile records in the data stream.

5 The user can optionally specify a message key and a program message queue as a source of data for a subfile of messages. The subfile modules receive a copy of the messages specified from the message queue and place these into the subfile.

6 The subfile is where all subfile information from the user buffer is stored. The subfile record remains in the subfile until the using program requests that it be displayed.

7 The source/sink data, which is a part of the request block, is where the subfile and console function managers build the console-dependent data stream that is to be transmitted to the console. The data stream contains all of the controls needed to display the user record as defined in the device file, and is built only when displaying the subfile via a put to the subfile control record.

8 The function manager work area is a part of the ODPCB (open data path control block). As the function manager builds the output data stream in the source/sink area, it also builds a user buffer image in the function manager work area. This work area contains as many buffer images as there are records with input capable fields on the console screen, except for subfile records. A single record buffer image is maintained for each subfile description, regardless of the number of records displayed.

9 The record, as seen in the user buffer, can optionally be sent to the job log as well as to the subfile. This is done independent of the subfile functions.

10 The I/O feedback area is updated at various times while the function manager is performing the put operation. Additional information, such as the relative record operated on, is put in the I/O feedback area for subfile functions.

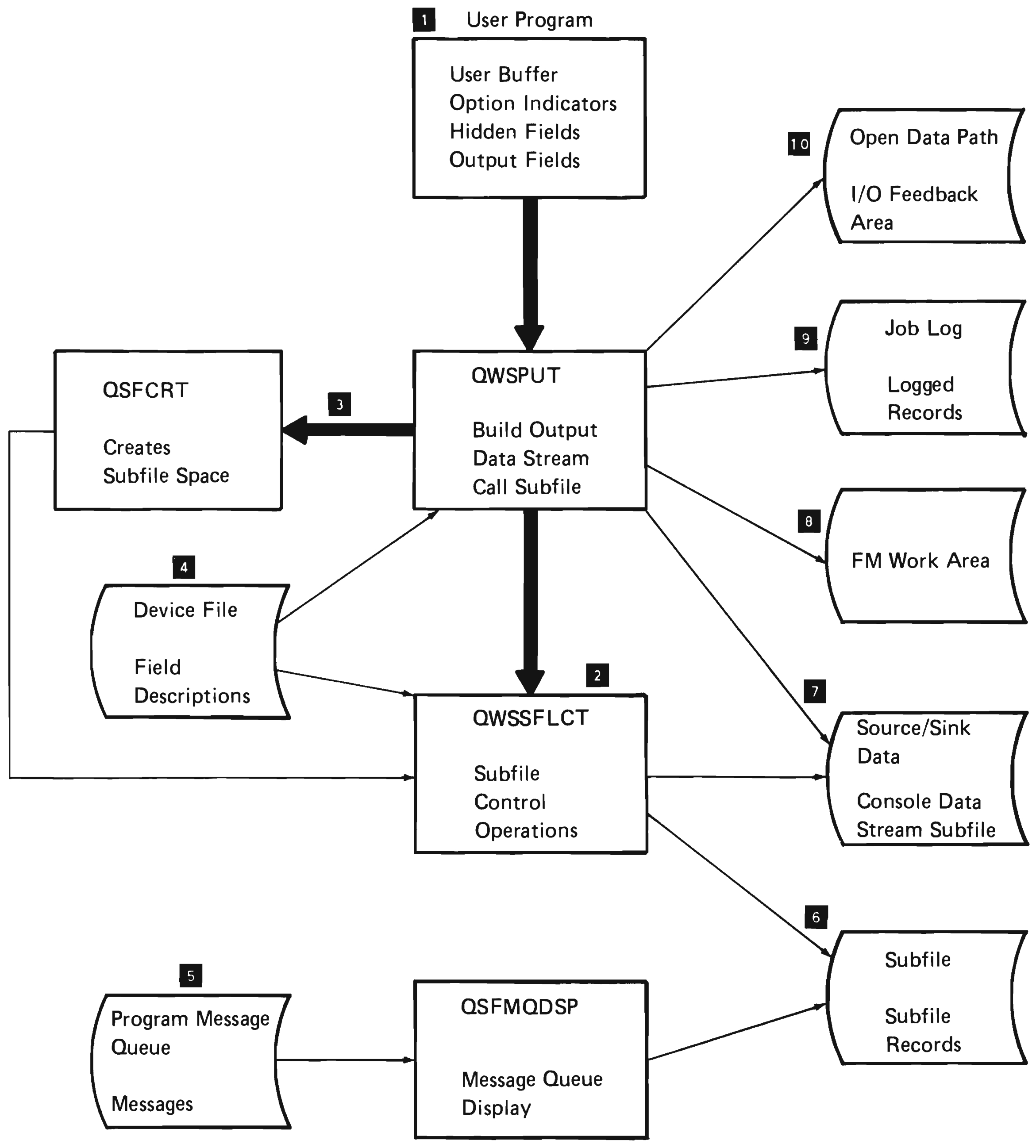


Figure WS-5. Put to Subfile Control Record (Data Flow)

Get Operation

Figure WS-6 and the following text describe a get operation.

1 Actual data does not come from the device file during a get operation, but record and field descriptions from the device file are used to process input information.

2 To initiate the read operation, a Read-Modified command is put in the source/sink data area and transmitted to the display. To satisfy the read, the display user must press a command key. If the record the display user wants was read and processed during a prior get operation, a Read-Modified command is not issued. Operation steps 3 through 7 are not performed, in such cases.

3 When the read operation is completed, all fields with the modified data tag set on are returned to the function manager in the same source/sink data area that contained the Read-Modified command.

4 If the field description specifies validity checks, such as range or list check, during field processing, QWSGET calls QCOVLFLD to perform those checks.

5 Input records can optionally be logged in the job log.

6 If subfile record fields are received, QWSGET alters its normal field process to handle the subfile records. When a subfile field is received, QWSGET retrieves the proper record from the subfile and puts it in the record save area of the function manager work area. All fields received for that subfile record are processed in the normal manner. When all fields are processed, the subfile record is returned to the subfile with the new data. This process is repeated for each subfile record that is modified by the display user.

7 When the function manager builds the output data stream in the source/sink data area, it also builds a user input buffer image in the function manager work area. These input buffer images are updated as QWSGET processes each field. At the completion of the get operation, all of the modified data on the display screen will be represented in the record save area of the function manager work area, in the subfile, or in both places.

8 The I/O feedback area is updated at various points in the function manager during the get operation.

9 For the normal field-level get operation, the user buffer contains response indicators followed by all of the input fields (including any hidden fields) as described in the device file. The data is moved from the record save area in the function manager work area to the user buffer.

During a nonfield-level get operation, the input data is moved directly from the source/sink data area to the user buffer.

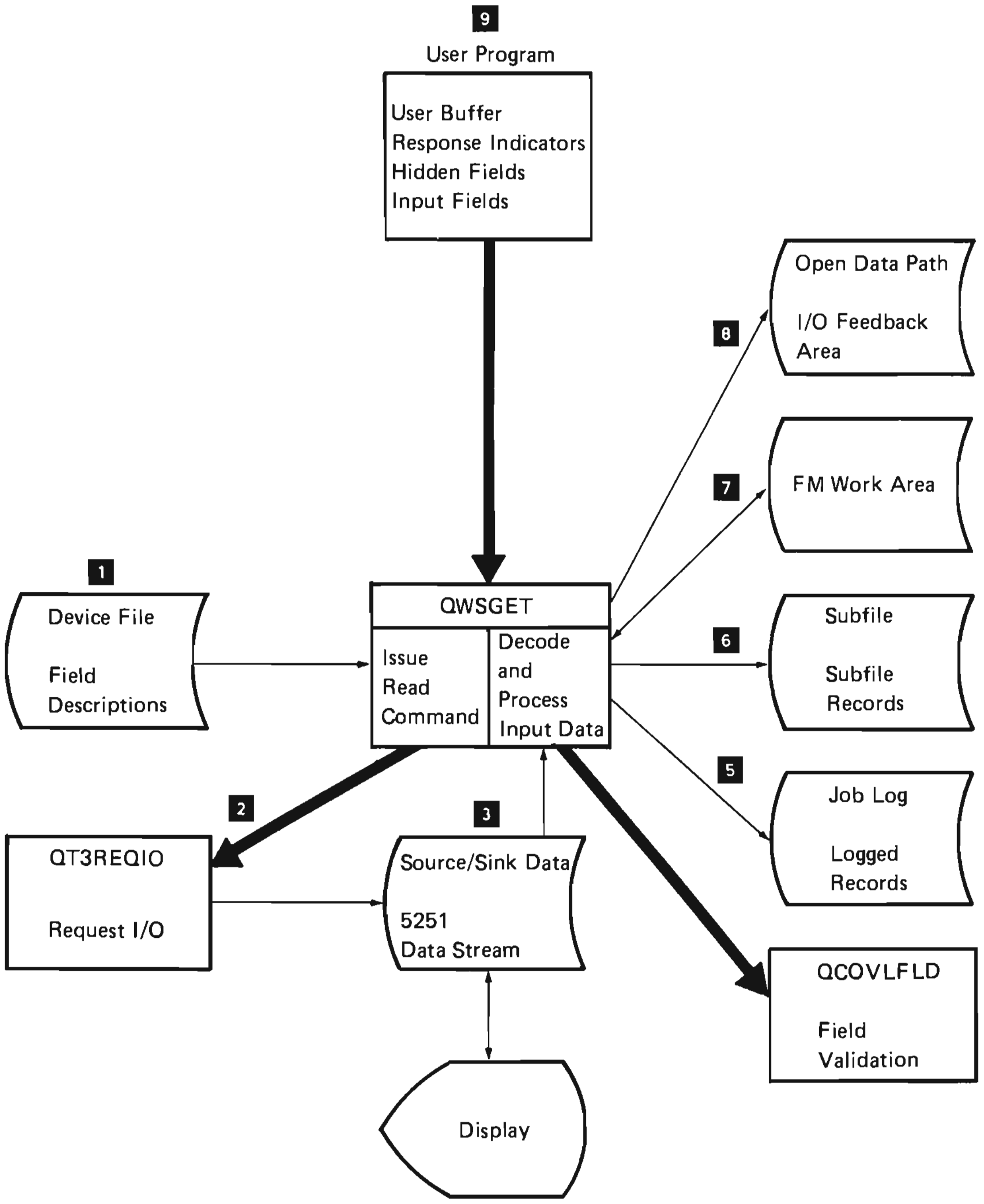


Figure WS-6. Get Operation

Get from Subfile Record (Data Flow)

Figure WS-7 and the following text describe a get from subfile operation.

1 Actual data does not come from the device file during a get operation, but record and field descriptions from the device file are used to process input information.

2 Input records can optionally be logged. The data is taken from the user buffer, and sent to the job log.

3 A get from subfile operation does not cause an I/O display operation. Instead, control is transferred to QSFGET to retrieve the requested record from the subfile.

4 QSFGET locates the correct record and moves it to the user buffer. All field processing and validating was done when the data was received from the display (see Figure WS-6, **6**). Because the records are stored in the subfile with the controls to redisplay them, the fields are put in the user buffer one field at a time.

5 Output-only fields are also returned to the user buffer when the record is a subfile record.

6 The I/O feedback area is updated at various points in the function manager while performing the get operation. In addition to the normal information, certain subfile information, such as relative record number returned, is also included in the I/O feedback area.

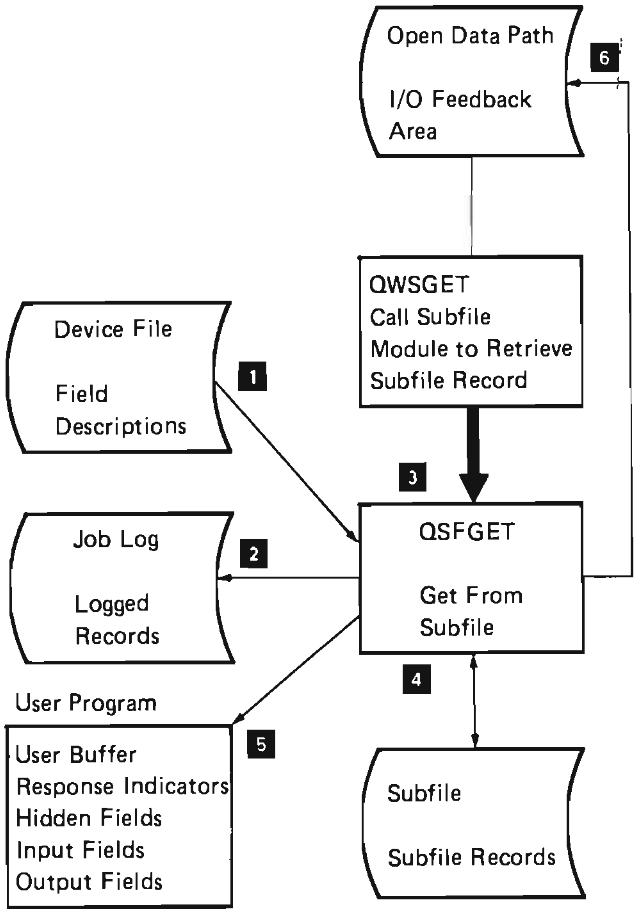


Figure WS-7. Get From Subfile Record (Data Flow)

This page is intentionally left blank.

Pass Option of the Suspend Module

The pass option of the QWSPEND indicates that the console user intends to pass across processes this display device and the unformatted data that was received on the last input request.

QWSPEND places a pointer to the last request block used by the get operation. A pointer to this new request block is placed in the LUD (logical unit description) associated space. This is how the actual request block is passed to the new process. By passing the logical unit description lock, the new process has addressability to the request block, which contains the unformatted data that was last read. The function manager close file always checks to see if a passed request block exists and, if one is found, destroys it along with the regular file request block.

Get (or Put-Get) Nowait Function

The get (or put-get) nowait request involves functions by several console function manager modules. Figure WS-8 and the following text describe the get (or put-get) nowait function.

1 The using program requests a get (or put-get) nowait operation.

2 QWSGET (or QWSPUT) processes the user request similar to a wait request, except that, before calling the QT3 REQIO module of SNA-T3 to do the request I/Os, it indicates in the request block that this is a nowait request. When QT3REQIO returns, QWSGET (or QWSPUT) returns to its caller.

3 QT3REQIO recognizes the nowait request and performs the REQIO instruction but does not wait for the request to complete nor does it do the dequeue of that request. Instead, QT3REQIO returns to its caller.

4 The machine issues the request I/O to the display as usual.

5 Up to this step, each module has processed the user request and returned to its caller. No module has waited for a response from the console device. The invocation stack contains only the using program.

6 When the operator responds to the get or the console device acknowledges the put, the machine enqueues the corresponding request block on the machine interface response queue in the normal manner. Because the nowait flag is set on in the request block, the machine signals the REQIO Complete event.

7 The REQIO complete event invokes the nowait event handler module, QWSMEEH, which does a dequeue of the completed request block.

8 QWSMEEH calls QT3REQIO to process the dequeued request block.

9 If the request was a get nowait, QWSMEEH calls QWSGET to process the input data. QWSGET recognizes the event handler call and processes just the user input data. If the operator requested a function-manager function, such as print or help text, QWSGET handles the request and reissues the nowait request.

10 When control is returned to QWSMEEH from QT3REQIO, it checks to see if valid data was entered. If valid data was entered, the data available event is signaled to the user. In an operator requested function, the nowait event handler module will not see any valid operator data and QWSMEEH terminates without signaling the data available event (flow returns to 3).

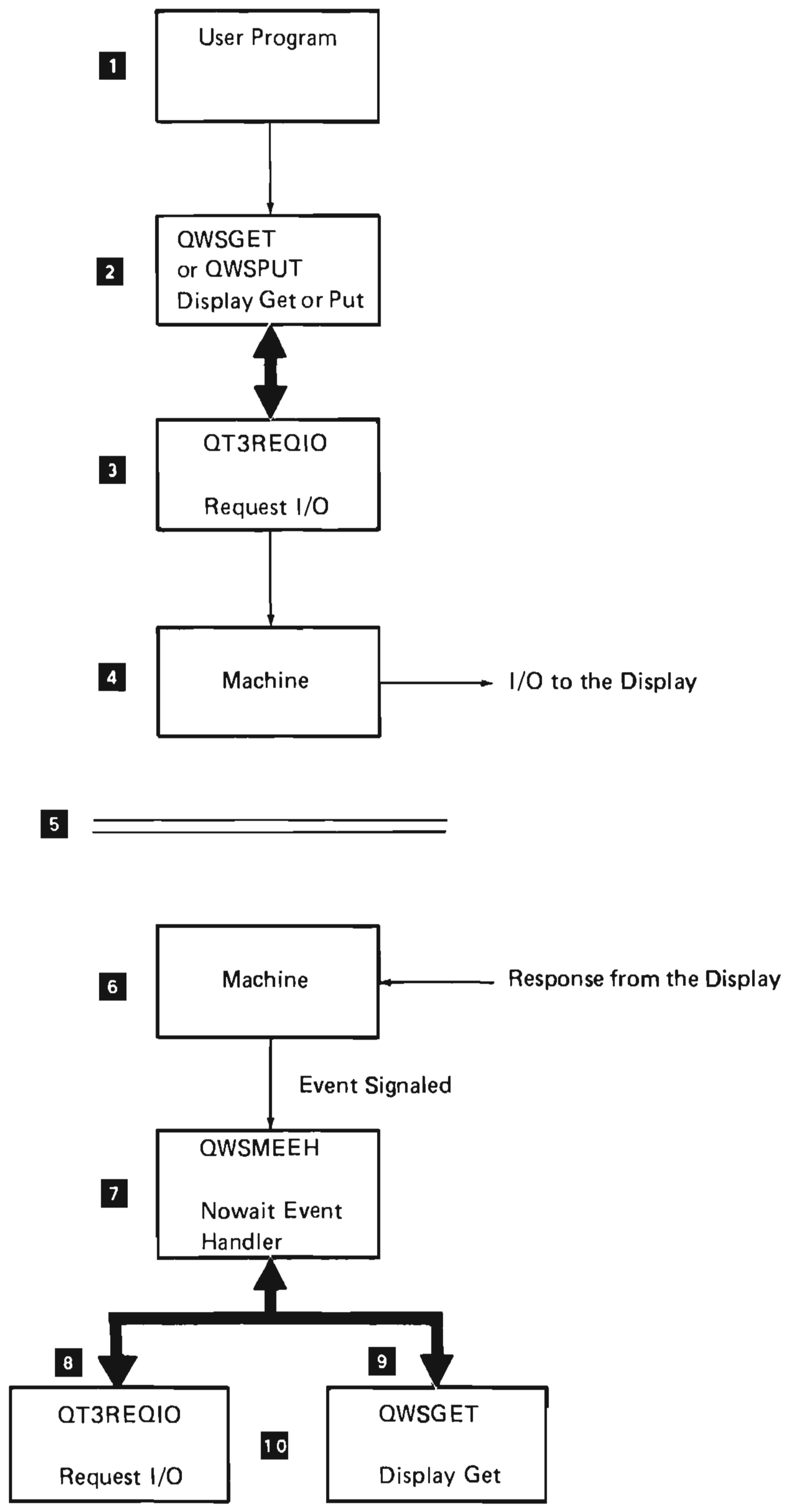
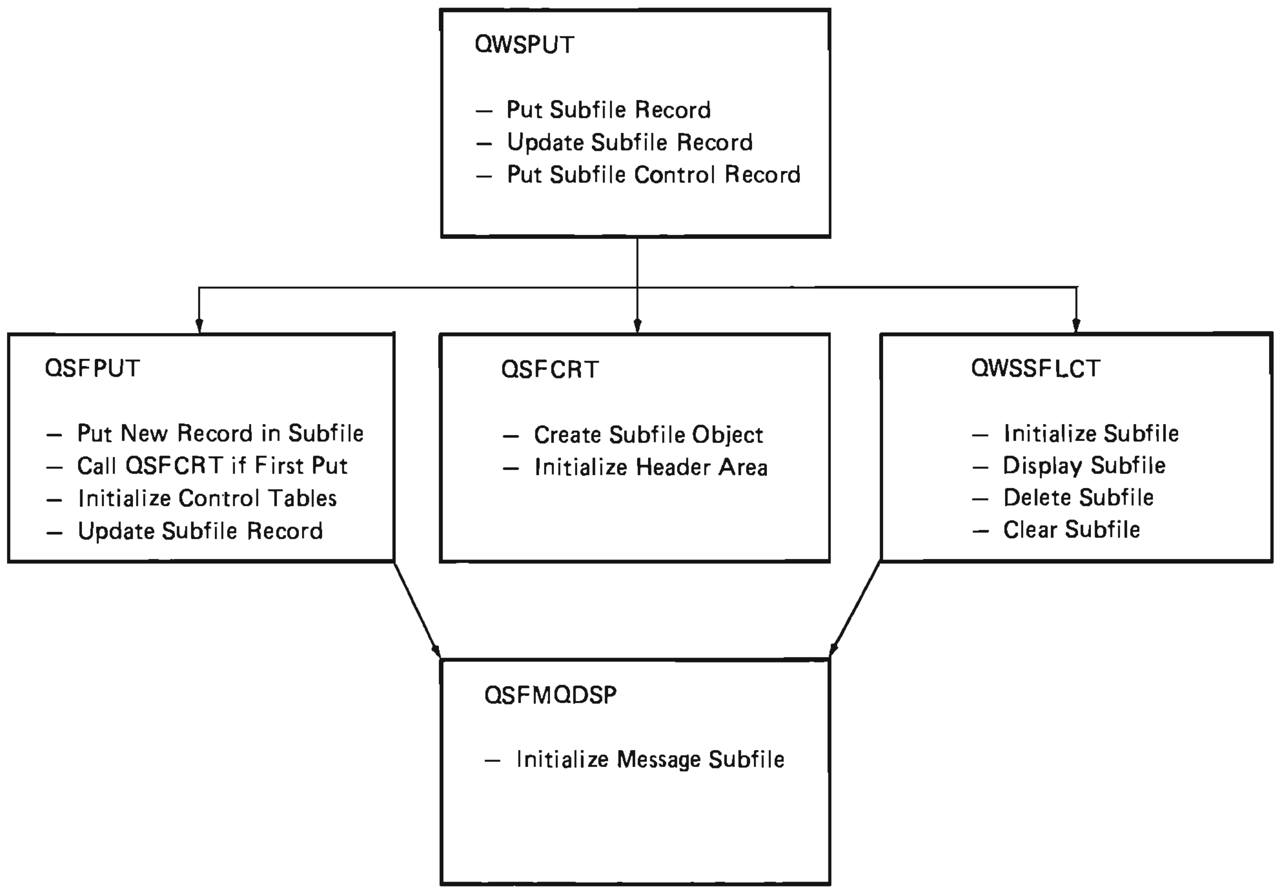


Figure WS-8. Nowait Function

Subfile Record (Module Flow)

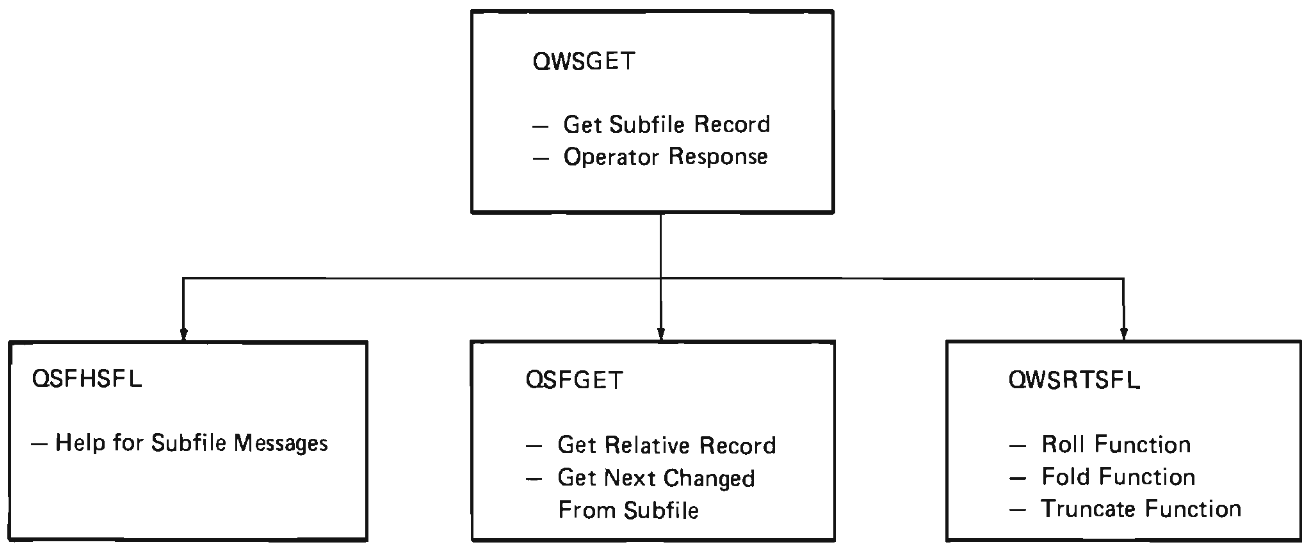
Figures WS-9 and WS-10 show the module flow for a put to subfile and a get from subfile operation.



Notes:

1. QSFPUT is invoked only for operations directed at the subfile record.
2. QWSSFLCT is invoked only for control record operations.

Figure WS-9. Put to Subfile Record (Module Flow)



Notes:

1. QSFHSFL is called only if the operator puts the cursor into a displayed subfile of messages and presses the Help key.
2. QWSRTSFL is called only if a subfile is displayed and the CA/CF key is pressed for fold/truncate or roll.

Figure WS-10. Get from Subfile Record (Module Flow)

I/O Error Flow

I/O errors are detected by the T3 component and processed by the 5251 function manager. The process consists of mapping the I/O error to a CPF message ID and sending the message to the program requesting the display data management service (see Figure WS-1). The user may monitor for these I/O error messages and perform local clean-up and recovery. If the user does not monitor for an I/O error message, the default action for most errors is to close the file.

The 5251 function manager modules are separated into two classes for the purpose of error handling:

- File-oriented modules (open, close, get, and put)
- Device-oriented modules (suspend, restore, and device event handlers)

I/O errors are presented to the user differently depending on whether the error is detected by a file-oriented module or a device-oriented module. The reason for the difference is that device-oriented modules do not have access to the file. File clean-up is therefore impossible for the device-oriented modules.

Figure WS-11 shows the I/O error processing flow for an error detected by a file-oriented module. Figure WS-12 shows the I/O error processing flow for an error detected by a device-oriented module.

Figure WS-11 describes the message and control flow (for an I/O or other request I/O detected error) when detected by a file-oriented module (open, close, get, put, or put-get).

1 The file-oriented function manager may be called by another CPF module or by a high-level language program.

2 The function manager module receives control and sets up message monitors for CPF5502 and CPF5601. During the course of its processing, the function manager module calls QT3 REQIO.

3 QT3 REQIO detects an abnormal condition and sends one or more diagnostic messages to its own program message queue. When QT3 REQIO finishes its processing it sends escape message CPF5502 to its caller. This causes QWSERROR to be invoked.

4 QWSERROR is invoked by exception CPF5502. QWSERROR processes the first condition detected by QT3REQIO, which is an error processing function and not an error recovery function. Only the first abnormal condition is processed by this module. All unprocessed conditions result in a message sent to the program message queue, and will be displayed in the job's message log.

The error routine receives and deletes the first diagnostic message from QT3REQIO's message queue. The information from this message is decoded, and a unique message ID is assigned. The device may be marked unusable, or other exception dependent processing done.

The message ID and message type are used to build a CPF5601 escape message, which is sent to the function manager module.

5 The CPF5601 internal exception handler in the function manager module uses the CPF message ID and message type to send the I/O error message to the display function manager user. The user may handle the error and continue processing, or ignore the error. If the error is ignored, the default action is taken (generally consisting of closing the file) and a function check is signaled to the user.

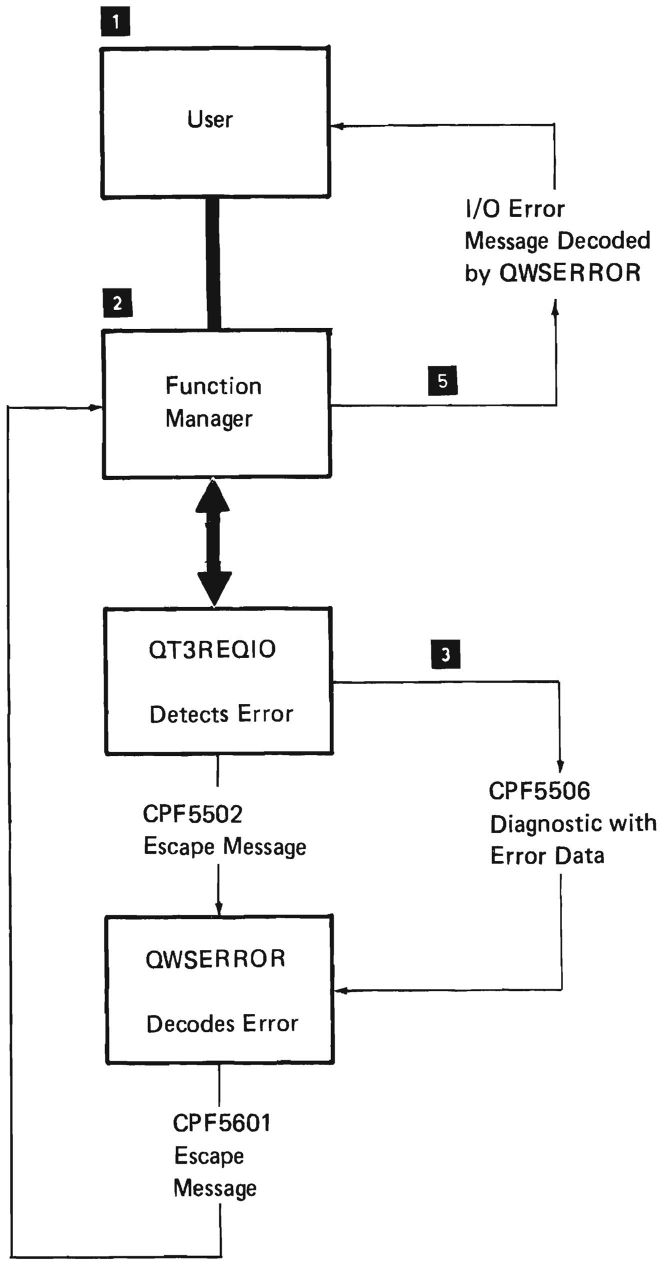


Figure WS-11. I/O Error Processing Flow–Errors Detected by a File-Oriented Module

Figure WS-12 describes the initial message and control flow (for an I/O or other REQIO-detected error) when detected by a device-oriented module (suspend, restore, or device event handlers).

1 The device-oriented function manager module may be invoked by the System/38, as in the case of device event handlers, or may be called by another CPF module.

2 The function manager module receives control and sets up message monitors for CPF5502 and CPF5601. During the course of its processing, the function manager module calls QT3 REQIO.

3 QT3 REQIO detects an abnormal condition and sends one or more diagnostic messages to its own program message queue. When QT3 REQIO finishes its processing, it sends notify message CPF5502 to its caller. This causes QWSERROR to be invoked.

4 QWSERROR is invoked by exception CPF5502. QWSERROR processes the first condition detected by the QT3REQIO, which is an error processing function and not an error recovery function. Only the first abnormal condition is processed by this module. All unprocessed conditions result in a message sent to the program message queue, and will be displayed in the job's message log.

The error routine receives and deletes the first diagnostic message from QT3REQIO's message queue. The information from this message is decoded, and a unique message ID is assigned. The device may be marked unusable, or other exception-dependent processing may be done.

The message ID and message type are used to build a CPF5503 diagnostic message containing the exception data. This diagnostic message is sent to the function manager module. The CPF5601 escape message contains a pointer to the CPF5503 diagnostic message and invokes an internal exception handler in the function manager module.

5 The CPF5601 internal exception handler in the function manager module sets the pointer to the CPF5503 diagnostic message describing the error into the function manager work area, where it may be retrieved later. Clean-up is done and a request failed notify message (not shown) is sent if the function manager module is not an event handler.

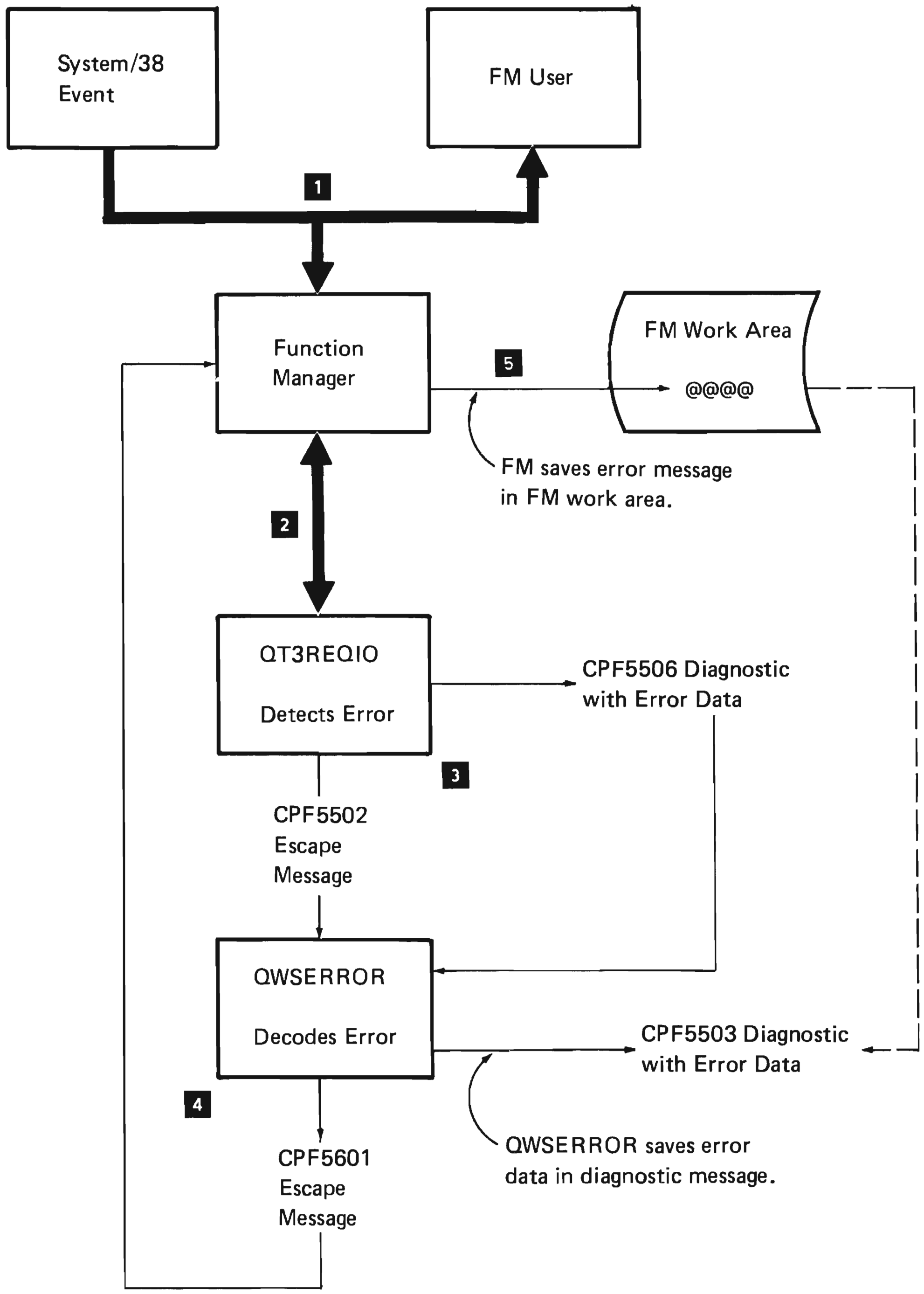

System/38 Event
FM User
1
Function Manager
5
FM Work Area
@@@@
FM saves error message in FM work area.
2
QT3REQIO
Detects Error
CPF5506 Diagnostic with Error Data
3
CPF5502 Escape Message
QWSERROR
Decodes Error
CPF5503 Diagnostic with Error Data
4
QWSERROR saves error data in diagnostic message.
CPF5601 Escape Message

Figure WS-12. I/O Error Processing Flow–Errors Detected by a Device-Oriented Module

Figure WS-13 describes the final message and control flow (error notification) for an I/O or other module-detected error when detected by a device-oriented module (suspend, restore, or device event handlers).

1 The file-oriented function manager module may be invoked by another CPF module or by a high-level language program. This is the first invocation of a file-oriented function manager module after an error has been detected.

2 The function manager module detects that an error condition has occurred, by checking for an error message saved in the function manager work area.

3 The function manager module retrieves the error message saved in the function manager work area and uses the exception data from that error message to build a *previous error* message, which is sent to the display function manager user. The user may either handle the error and continue processing or ignore the error. If the error is ignored, the default action is taken (generally consisting of closing the file) and a function check is signaled to the user.

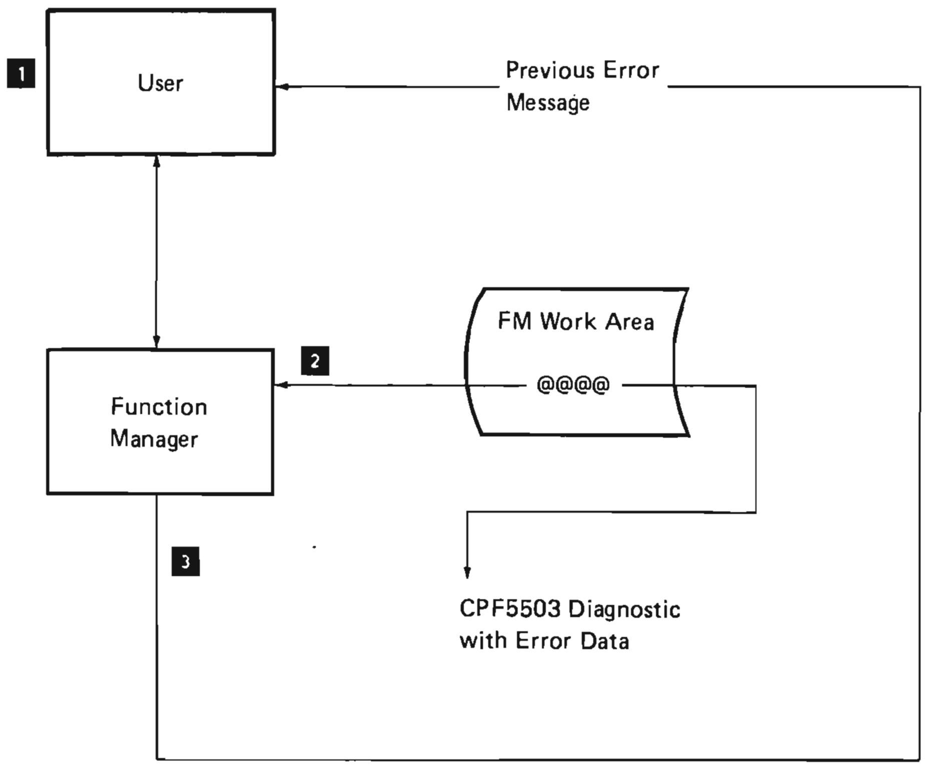


Figure WS-13. I/O Error Notification–Errors Detected by a Device-Oriented Module

INTRODUCTION

The work monitor component of the CPF (control program facility) provides support for initiating, controlling, and terminating user jobs in the system.

The functions provided by the work monitor are:

- Subsystem startup, control and termination
- Job initiation, control and termination
- Work station support
- Work management support of APPC (advanced program-to-program communications)
- Create job structure support
- Batch job creation, routing control
- System request support
- Attention key support
- Group job support

GENERAL OVERVIEW

Work Monitor Modules

Each module listed below performs a function.

Subsystem Startup Modules

-->QWTMCACE–Allocate Communications Entries: This module requests allocation of peer devices from the logical unit services (LUS) process to a subsystem.

-->QWTMCAJE–Analyze Job Entry: This module analyzes and processes both auto start job entries and the console job after the start CPF processing.

-->QWTMCDFT–Create Monitor Device and File Table: This module creates data structures to support work stations and peer devices.

-->QWTMCQAL–Allocate Job Queue: This module attempts to allocate job queues to a subsystem.

-->QWTMCSTP–Subsystem Startup: This module creates the subsystem environment.

-->QWTMEAST–Allocate Storage Event: This module requests allocation of storage pools from the arbiter process to a subsystem.

-->QWTMEJQA–Job Queue Available Event: This module allocates a job queue to a subsystem when it becomes available.

-->QWTMESTA–Storage Allocated Event: This module allocates storage pools to a subsystem.

-->QWTMMALM–Peer Device Allocated: This module allocates a peer device to a subsystem.

Subsystem Control Modules

-->QWTMESBC–Subsystem Control Event: This module handles all subsystem control requests and invokes subsystem control functions.

-->QWTMMCSD–Change Active Subsystem: This module changes an active subsystem environment.

Subsystem Termination Modules

-->QWTAMABT–Abnormal Subsystem: This module handles subsystem termination under abnormal conditions.

-->QWTCLNUP–Abnormal System: This module handles subsystem and user job termination after an abnormal system termination.

-->QWTMCDCE–Deallocate Communications Entries: This module deallocates peer devices from a subsystem.

-->QWTMCQDA–Deallocate Job Queues: This module allocates job queues from a subsystem.

-->QWTMCSTD–Subsystem Shutdown: This module performs the last functions before a subsystem terminates.

-->QWTMMTRS–Subsystem Termination: This module performs the initial termination function in a subsystem.

Job Initiation Modules

-->QWTMCATT–Peer Device Evoke Request: This module processes any evoke requests received on a peer device allocated to a subsystem. QWTMCATT also sets up any APPC batch jobs for initiation.

-->QWTMCMNL–Subsystem Mainline Program: This module initiates user processes.

-->QWTMEATR–525X Test Request Event: This module handles a test request on a 525X work station and sets up a test request job for initiation.

-->QWTMEJIN–Job Queue Job Initiation: This module processes jobs on job queues and sets up batch jobs for initiation.

-->QWTMERQD–Routing Data Available Event: This module processes data from the manual routing prompt.

-->QWTMESGN–Sign-on Data Available Event: This module processes data from the sign-on prompt and sets up an interactive job for initiation.

-->QWTMMERH–Job Initiation Error: This module handles errors encountered when a subsystem attempts to initiate a user process.

-->QWTPCRJA–Retrieve Job Attributes: This module retrieves the attributes of a user job.

-->QWTPIIPP–Process Initiation Phase Program: This module completes initialization of a subsystem or user process.

-->QWTPIPPP–Evoke Program Parameter: This module handles the parameters for the user program of an APPC job.

Job Control Modules

Each module listed below processes the command listed next to the module.

-->QWTCCCHJ–Change Job (CHGJOB command)

-->QWTCCCNJ–Cancel Job (CNLJOB command)

-->QWTCCHDJ–Hold Job (HLDJOB command)

-->QWTCCJOB–Job (JOB command)

-->QWTCCRLJ–Release Job (RLSJOB command)

-->QWTCCRRJ–Reroute Job (RRTJOB command)

-->QWTCCRTN–Return (RETURN command)

-->QWTCCSBJ–Submit Job (SBMJOB command)

-->QWTCCTFJ–Transfer Job (TFRJOB command)

-->QWTCCTBJ–Transfer Batch Job (TFRBCHJOB command)

Each module listed below performs a function.

-->QWTMMCHJ–Change Job: This module performs the change function for an active job.

-->QWTMMCNJ–Cancel Job: This module performs the cancel function for an active job.

-->QWTMMHDJ–Hold job: This module performs the hold function for n active job.

-->QWTMMRLJ–Release Job: This module performs the release function for an active job that is on hold.

-->QWTPECTL–Process Control Event: This module handles all process control requests and invokes process control functions.

-->QWTPMCNJ–Controlled Cancel Job: This module performs the controlled cancel function for an active job.

-->QWTSCSBJ–SBMJOB/JOB Command Processor: This module handles the common function of the SBMJOB and JOB command.

-->QWTSETME–Controlled Cancel Job Time-out: This module terminates a user job when the controlled cancel job timer expires.

Job Termination Modules

-->QWTMEOJ–End-of-Job: This module performs end-of-job functions in a subsystem or user job.

-->QWTMETRP–Process Termination: This module handles user process termination in a subsystem.

-->QWTMMBJT–Batch Job Termination: This module handles job termination for all batch jobs.

-->QWTMMIJT–Interactive Job Termination: This module handles job termination for all interactive jobs.

-->QWTMMTJT–Transfer Job Termination: This module handles job termination for all transfer jobs.

-->QWTPITPP–Process Termination Phase Program: This module performs termination functions for a subsystem or user process.

Work Station Support Modules

-->QWTCCSTA–Set Attention Program (SETATNPGM) CPP: This module processes the SETATNPGM command.

-->QWTSEATN–Attention Key Event Handler: This module calls the user-defined attention key handler when the ATTN key is pressed.

-->QWTSCOPR–Attention Key Scope Handling Routine: This module cleans up attention key handling when an invocation using the SETATNPGM command has completed.

-->QWTSCPSH–Push Attention Key Routine: This module pushes an entry onto a stack that controls attention key handlers.

-->QWTSCPOP–Pop Attention Key Routine: This module pops an entry off of a stack that controls attention key handlers.

-->QWTAESRQ–Work Station: This module indicates if the system request or test request key has been pressed or if there is unsolicited work station data.

-->QWTCCSRQ–Transfer to Secondary Job (TFRSECJOB command): This module processes the Transfer to Secondary Job command.

-->QWTMCDSP–Sign-On Prompt Display: This module displays the sign-on prompt at a work station or at the console.

-->QWTMCDVX–Work Station Device Error: This module handles work station device errors.

-->QWTMCERP–Work Station Error Recovery: This module handles a work station that has recovered from an error.

-->QWTMEOBD–Obtain Work Station: This module reassigns a work station from one subsystem to another.

-->QWTMEPDA–Work Station Allocation: This module allocates a work station to a subsystem when it becomes available.

-->QWTMESRQ–Process Suspended: This module performs the transfer to the secondary job function (option 1 on the system request menu).

-->QWTPCSRQ–Transfer to Secondary Job: This module requests the transfer to the secondary job function.

-->QWTPMSRQ–System Request: This module processes a system request in a user job.

Advanced Program-to-Program Support Modules

-->QWTLCALC–Peer Device Contact: This module processes a peer device when a remote system is contacted.

-->QWTLCAMS–Allocate Peer Device to Subsystem: This module allocates a peer device to a subsystem.

-->QWTLCATT–Peer Device Evoke Request: This module processes evoke requests that are received on the peer devices allocated to the logical unit services process.

-->QWTLCRDT–Create Peer Device Table: This module creates a data structure for peer device allocation to subsystems.

-->QWTLCVOF–Peer Device Vary Off: This module processes a peer device when it is varied off.

-->QWTLCVON–Peer Device Vary On: This module processes a peer device when it is varied on.

-->QWTLEARM–Evoke Request Maximum Exceeded: This module handles evoke requests when no conversations are available.

-->QWTLEDRQ–Peer Device Request: This module processes a peer device request from a subsystem.

-->QWTLEOLK–Peer Device Allocated: This module processes a peer device when it is returned from a subsystem to the logical unit services process.

Create Temporary Job Structure Support Module

-->QWTAMCJS–Create Temporary Job Structure: This module creates temporary job structures when the system is started and after all the temporary job structures are used.

Subsystem Functions

Subsystem Startup

A subsystem is started when the controlling subsystem is automatically started during start CPF processing or when the Start Subsystem (STRSBS) command is entered. In either case, the system arbiter starts a process in which the subsystem modules will execute. The following functions are performed during subsystem startup:

- The initiation phase module (QWTPIIPP) for the process will perform common CPF process initialization functions.

- The subsystem problem phase program (QWTMCSTP) will perform specific initialization functions and set up the subsystem environment as specified in the subsystem description. The following functions are performed (unless the subsystem is started from the restricted state):
  - Create and initialize data structures used by the subsystem modules.
  - Request to allocate the storage pools defined in the subsystem description. When a storage pool is allocated, the allocated module (QWTMCSTA) is invoked to complete allocation.
  - To initiate autostart jobs, the analyze job entry module (QWTMCAJE) is invoked to process each autostart job specified in the subsystem description. QWTMCAJE will initialize a job structure for the autostart job and notify the subsystem when the job is ready for initiation.
  - To support work stations, the create work station device and file table module (QWTMCDFT) is invoked to create data structures to support the work stations and the sign-on and manual-routing display files. For each work station specified as *SIGNON in the work station entry in the subsystem description, the device description associated with the work station is allocated to the subsystem and the sign-on display file appears. If a work station cannot be allocated during subsystem startup, the subsystem will allocate the device when it becomes available.

- To support job queues, a data structure is created. The job queue allocation module (QWTMCQAL) is invoked to allocate each job queue specified in the subsystem description. If a job queue cannot be allocated, the subsystem will allocate the job queue when it becomes available. The job queue available module (QWTMEJQA) is invoked to complete allocation.
- To support peer devices, QWTMCDFT is invoked to create a data structure to support peer devices. The peer device allocation module (QWTMCACE) is invoked to request allocation of each peer device specified in the subsystem description. When a peer device is allocated, the peer device allocated module (QWTMMALM) is invoked to complete allocation.

Subsystem Control

The initial subsystem environment as defined in the subsystem description (pool size, MPL classes and number of maximum active jobs) can be changed with the change subsystem description command (CHGSBSD). When this command is invoked, the subsystem control module (QWTMESBC) is invoked in the subsystem to be changed. This module can change the subsystem environment. QWTMESBC invokes the change-active subsystem module (QWTMMCSD) to perform the requested changes.

Subsystem Termination

There are three commands to terminate a subsystem: the power down system command (PWRDWNSYS), the terminate CPF command (TRMCPF) and the terminate subsystem command (TRMSBS). These commands will cause the subsystem termination module (QWTMMTRS) to be invoked in the subsystem. All work stations allocated to the subsystem are deallocated. All job queues allocated to the subsystem are deallocated by the deallocate job queue module (QWTMCDQA). All peer devices allocated to the subsystem are deallocated by the deallocate communications entries module (QWTMCDCE). All jobs active in the subsystem are either terminated (if the subsystem termination is immediate) or notified of termination (if the termination is controlled). If there are no active jobs in the subsystem or when all jobs have terminated, the subsystem shutdown module (QWTMCSTD) is invoked.

If the subsystem is terminating to the restricted state, QWTMCSTD deallocates the subsystem storage pools and MPL classes so that the remaining subsystem data structures are not destroyed. When this module completes, the termination phase program (QWTPITPP) is invoked to complete the subsystem process termination.

Abnormal Subsystem Termination

If a subsystem process abnormally terminates (cannot perform the normal termination function as described above) the abnormal subsystem termination module (QWTAMABT) is invoked in the system arbiter process. This module terminates any user jobs that are still active and waiting to be terminated.

Cleanup After Next IMPL

The subsystem cleanup module (QWTCLNUP) is invoked at the next IMPL for each job that was active when the system terminated. If the log for each job has not been written, this module invokes the end-of-job cleanup module (QWTMCEOJ). QWTMCEOJ performs normal end-of-job functions other than spooling the job log to an output queue, which is done later in start CPF processing.

Job Functions

Job Initiation

Work initiation describes the functions performed by the subsystem prior to initiating the processes that perform the user work. These functions include the processing of autostart job entries contained in a subsystem description, the processing of sign-on, test and evoke requests received by a subsystem, and the selection of batch jobs from a job queue. The main function performed for each type of job is to allocate and initialize a job structure for the user job. After the job structures are initialized the subsystem is told the job is ready for initiation.

- The analyze job entry module (QWTMCAJE) is invoked to process each autostart job specified in the subsystem description.

- The sign-on data available module (QWTMESGN) and, optionally, the manual routing data available module (QWTMERQD) are invoked to process work station jobs.

- The test request module (QWTMEATR) is invoked to process test request jobs.

- The attach manager module (QWTMCATT) is invoked to process APPC batch jobs. If the evoke request specifies that parameters will be passed to the user's problem phase program, QWTMCATT sets up QWTPIPPP as the problem phase program for the job. QWTPIPPP receives the parameters for the problem phase program for the evoke request and invokes that program with the parameters.

- The job queue job initiation module (QWTMEJIN) is invoked to process batch and interactive jobs on a job queue. A job can be placed on a job queue due to the submit job command (SBMJOB), the job command (//JOB), the transfer job command (TFRJOB) or the transfer batch job command (TFRBCHJOB).

Once a job structure is initialized, the subsystem is told that a job is ready to be initiated. The process initiation module (QWTMCMNL) is invoked in the subsystem and QWTMCMNL uses the routing data specified with the job to find the appropriate routing entry for the user job. (The routing entry is listed in the routing table defined in the subsystem description.) The routing entry data tells QWTMCMNL to initialize the process definition template and set up the process initiation phase program (QWTPIIPP) and termination phase program (QWTPITPP) to run before and after the user's problem phase program. The process is initiated if no errors have been detected. If an error is detected, the subsystem error handling module (QWTMMERH) is invoked to recover from the error.

When the process is initiated, QWTPIIPP is invoked to initialize the process for user work. If this is a batch job that was submitted to a job queue by the Submit Job (SBMJOB) command, the retrieve job attributes module (QWTPCRJA) is invoked to get the current job attributes for an information message that is sent to the job log. After QWTPIIPP completes, the user's problem phase program is invoked.

Job Control

Job control is performed by the Hold Job command (HLDJOB), Release Job command (RLSJOB), Change Job command (CHGJOB), Change Accounting Code command (CHGACGCDE) and Cancel Job command (CNLJOB). The following functions are performed by each job control command:

- When the HLDJOB command is invoked, the hold job command processing program (QWTCCHDJ) is invoked in the user's process. QWTCCHDJ confirms that the user can hold the job and determines the state of the job to be held. A user can request QWTCCHJD to hold the spooled output. If the job is queued, QWTCCHDJ holds the job. If the job is active, QWTCCHDJ tells the subsystem to hold the job and invokes the subsystem control module (QWTMESBC) in the subsystem. QWTMESBC invokes the hold job module (QWTMMHDJ) to hold the active job in the subsystem. QWTMMHDJ confirms that the job is in the right state to be held and suspends the job process.

- When the RLSJOB command is invoked, the release job command processing program (QWTCCRLJ) is invoked in the user's process. QWTCCRLJ confirms that the user can release the job and determines the state of the job to be released. QWTCCRLJ releases the spooled output if it is on hold. If the job is queued, QWTCCRLJ releases the job. If the job is active, QWTCCRLJ tells the subsystem to release the job and invokes the subsystem control module (QWTMESBC) in the subsystem. QWTMESBC invokes the release job module (QWTMMRLJ) to release the job that is on hold in the subsystem. QWTMMRLJ confirms that the job is in the right state to be released and resumes the job process.

- When the CHGJOB command is invoked, the change job command processing program (QWTCCCHJ) is invoked in the user process belonging to the user who requested the change. QWTCCCHJ confirms that the user can change the job and determines the state of the job to be changed. If the job is queued or if it is the user's job, QWTCCCHJ changes the requested job attributes. If the job is active and is not the user's job, QWTCCCHJ tells the subsystem to change the job and invokes the subsystem control module (QWTMESBC) in the subsystem. QWTMESBC invokes the change job module (QWTMMCHJ) to change an active job in the subsystem. QWTMMCHJ confirms that the job is in the right state to be changed and changes the requested job attributes.

  QWTCCCHJ is also used by the change accounting code command (CHGACGCDE), which performs the same function as CHGJOB except the accounting data is the only data changed. The current accounting data is journaled.

- When the CNLJOB command is invoked, the cancel job command processing program (QWTCCCNJ) is invoked in the user's process. QWTCCCNJ confirms that the user can cancel the job and determines the state of the job to be canceled. A user can request QWTCCCNJ to cancel the spooled output. If the job is queued, QWTCCCNJ cancels the job. If the job is active, QWTCCCNJ tells the subsystem to cancel the job and invokes the subsystem control module (QWTMESBC) in the subsystem. QWTMESBC invokes the cancel job module (QWTMMCNJ) to cancel the active job on the subsystem. QWTMMCNJ confirms that the job is in the right state to be canceled and the subsystem terminates the process. If the job is to be canceled with the controlled option, QWTMMCNJ tells the job that it is being canceled. This results in the process control module (QWTPECTL) being invoked in the user's process. QWTPECTL invokes the cancel job controlled module (QWTPMCNJ), which sets up a timer for the amount of time specified on the CNLJOB command. When the time expires, the timer module (QWTSETME) is invoked and terminates the process.

Batch Job Creation

A user can use the submit job command (SBMJOB) or the job command (//JOB) to create a batch job and place it on a job queue. The following functions are performed by each batch job creation command:

- When the SBMJOB command is invoked, the submit job command processing program (QWTCCSBJ) is invoked in the user's process. QWTCCSBJ confirms that the user can create a batch job and invokes the batch job creation module (QWTSCSBJ) to set up the job. QWTSCSBJ initializes a job structure and QWTCCSBJ places the job on a job queue.

- When the //JOB command is found in a file by a reader or by a SBMCRDJOB, SBMDBJOB or SBMDKTJOB command, the job command processing program (QWTCCJOB) is invoked. QWTCCJOB invokes the batch job creation module (QWTSCSBJ) to set up the job. QWTSCSBJ initializes a job structure for the job and QWTCCJOB tells the invoking function to place the job on a job queue.

Routing Control

The routing steps of a job can be controlled by the transfer job command (TFRJOB), the transfer batch job command (TFRBCHJOB), the reroute job command (RRTJOB) and the return command (RETURN). The following functions are performed by each routing control command:

- When the TFRJOB command is invoked, the transfer job command processing program (QWTCCTFJ) is invoked in the user's process. QWTCCTFJ confirms that the user can transfer to another job queue and that the transfer is made in the right environment. If the transfer is allowed, QWTCCTFJ terminates the user's process and invokes the process terminated module (QWTMETRP) to complete termination of the process in the subsystem. QWTMETRP invokes the transfer job termination module (QWTMMTJT) to place the job on a job queue.

- When the TFRBCHJOB command is invoked, the transfer batch job command processing program (QWTCCTBJ) in invoked in the user's process. QWTCCTBJ performs the same function as QWTCCTFJ except that specific job information is stored in a permanent object so that if the system terminates after transferring a batch job to a job queue, the job can be initiated following an IMPL. The TFRJOB command does not have this function.

- When the RRTJOB command is invoked, the reroute job command processing program (QWTCCRRJ) is invoked in the user's process. QWTCCRRJ confirms that the user can change routing steps within the subsystem. If the reroute is allowed, QWTCCRRJ terminates the user's process and invokes the process terminated module (QWTMETRP) to complete the termination of the process in the subsystem. QWTMETRP invokes the process initiation module (QWTMCMNL) in the subsystem to initiate the user's process again.

- When the RETURN command is invoked, the return job command processing program (QWTCCRTN) is invoked in the user's process. QWTCCRTN tells the command language request processing program (QCL) to end the current request. If QCL is the highest level invocation in the user's process, QCL will complete, the process will terminate, and the process terminated module (QWTMETRP) is invoked to complete the termination of the process in the subsystem. QWTMETRP causes the process initiation module (QWTMCMNL) to be invoked in the subsystem to initiate the user's process again.

Job Termination

When the user's problem phase program completes, the process termination phase program (QWTPITPP) is invoked to perform process clean-up functions. This includes closing all files that were opened by the process. If the process is terminating due to an end-of-job condition, the end-of-job clean-up module (QWTMCEOJ) is invoked. QWTMCEOJ performs functions that are specific to the end-of-job condition, which includes cleaning up the spool files and spooling the job log to an output queue.

The user process is completed when QWTPITPP has completed the clean-up functions. Next, the process terminated module (QWTMETRP) is invoked in the subsystem that initiated the process. QWTMETRP determines the type of job terminated and invokes the appropriate module to perform the specific process termination function. If a batch or autostart job was terminated, the batch job termination module (QWTMMBJT) is invoked. If the job type is interactive, the interactive job termination module (QWTMMIJT) is invoked. If the job terminated due to a Transfer Job command (TFRJOB or TFRBCHJOB), the transfer job termination module (QWTMMTJT) is invoked. QWTMETRP also handles job termination because of a Reroute command (RRTJOB) or a Return command (RETURN or CF1 key from highest invocation).

Work Station Support

- If a subsystem cannot allocate a work station at startup, the subsystem sends a request to allocate the work station when it becomes available. Once it is available, QWTMEPDA is invoked in the subsystem to handle work station allocation.

  If a work station is allocated to a subsystem and the sign-on prompt appears, the work station can be reallocated to another subsystem or user job using the QWTMEOBD module.

- If an error is found during a work station device function that involves the sign-on prompt or the manual routing prompt, the work station error handling module (QWTMCDVX) is invoked. QWTMCDVX determines the error type and takes appropriate action. If the error occurred because the work station was powered off, QWTMCDVX sets up a module (QWTMCERP) to be invoked if the work station is powered on. When the work station is powered on, QWTMCERP is invoked in the subsystem. QWTMCERP then invokes the display sign-on prompt module (QWTMCDSP) to display the sign-on prompt on the work station. (QWTMCDSP is also invoked from QWTMCSTP and QWTMEPDA when a work station is allocated to a subsystem and from QWTMESRQ for system request.)

System Request Support

- When the system request key is pressed at a work station, the work station module (QWTAESRQ) is invoked in the system arbiter. QWTAESRQ determines the job associated with the work station and tells the job about the system request. A module is invoked in the work station job (QWTPECTL) to handle the notification. QWTPECTL invokes the system request handler (QWTPMSRQ), which invokes the system request menu handler. If option 1 is selected from the menu, the menu handler invokes the transfer to secondary job functions (QWTPCSRQ). QWTPCSRQ suspends the process of the job that invoked the transfer to secondary job function and invokes the process suspended module (QWTMESRQ) in the subsystem. QWTMESRQ either displays the sign-on display on the work station, if no secondary job exists, or resumes the secondary job. The transfer to secondary job function can also be performed by invoking the transfer to secondary job command (TFRSECJOB). When the TFRSECJOB command is invoked, the transfer to a secondary job processing program (QWTCCSRQ) is invoked. QWTCCSRQ performs the same functions that are performed in QWTAESRQ, QWTPMSRQ, and QWTPCSRQ.

Attention Key Support

When the Set Attention Program (SETATNPGM) command is invoked, QWTCCSTA is invoked in the user's process. This module locates the user specified attention key handling routine and verifies the user's authority to call that program. It then uses the attention key routines to set up attention key handling as requested so that the correct program will be invoked when the attention key is pressed.

When the ATTN key is pressed, the attention key event handling routine (QWTSEATN) is invoked in the user's process. This routine invokes the user specified attention key handling program and also handles any errors that may occur.

When an invocation that uses the SETATNPGM command returns, the attention key scope handling routine is invoked. This routine invalidates attention handling set up by the invocation, and restores any attention handling that may have been used in the previous invocation.

Group Job Support

The following text describes the support for group jobs for the various CL commands.

- Change Group Attributes (CHGGRPA) Command–When the CHGGRPA command is invoked, the change group attributes command processing program (QWTCCCHG) is invoked in the user's process. This module verifies that the user can change the attributes of the job. If the interactive job is to be changed into a group job, changed back into a non-group job, or if the group job text is to be changed, QWTCCCHG notifies the job's subsystem that the job is to be changed. This results in the subsystem control module (QWTMESBC) being invoked in the job's subsystem. QWTMESBC determines that the function is to change the group attributes of a job in the subsystem and invokes the change group attributes module (QWTMMCHG) to perform the requested function. All other attributes are changed by QWTCCCHG.
- Retrieve Group Attributes (RTVGRPA) Command–When the RTVGRPA command is invoked, the retrieve group attributes command processing program (QWTCCRVG) is invoked in the user's process. This module verifies that the user can retrieve the attributes of the group job. If the list of jobs in the user's group is to be retrieved, QWTCCRVG notifies the job's subsystem that the job's group job list is to be retrieved. This results in the subsystem control module (QWTMESBC) being invoked in the job's subsystem. QWTMESBC determines that the function is to retrieve the list of group jobs for a group job and invokes the retrieve group attributes module (QWTMMRVG) to perform the requested function. All other attributes are retrieved by QWTCCRVG.
- Transfer Group Job (TFRGRPJOB) Command–When the TFRGRPJOB is invoked, the transfer to group job command processing program (QWTCCTFG) is invoked in the user's process. This module verifies that the user can transfer to a group job. QWTCCTFG suspends the process of the job that issued the transfer to group job command, which causes the process suspended module (QWTMESRQ) to be invoked in the job's subsystem. If the specified group job does not exist, this module allocates and initializes a job structure for a group job and notifies the subsystem that the job is ready to be initiated. If the group job does exist, this module resumes the specified group job.

- Terminate Group Job (TRMGRPJOB) Command–When the TRMGRPJOB command is invoked, the terminate command processing program (QWTCCTRG) is invoked in the user's process. This module verifies that the user can terminate a group job. QWTCCTRG notifies the job's subsystem that a group job is to be terminated. This results in the subsystem control module (QWTMESBC) being invoked in the job's subsystem. QWTMESBC determines that the function is to terminate a group job in the subsystem and invokes the terminate group job module (QWTMMTRG) to perform the requested function.

Advanced Program-to-Program Communications Support

The work monitor support for APPC involves the reallocation of APPC peer devices from a subsystem with communications work entries and involves initiating jobs upon receipt of an evoke request from a peer device. The modules involved in reallocating peer devices are as follows:

- At the IMPL for the System/38, a system process is initiated to handle peer devices. This process is called the logical unit services (LUS) process. As part of process initialization, the create peer device table module (QWTLCRDT) is invoked to create a data structure to handle the reallocation of peer devices.

- When a peer device is varied on, CPF modules are invoked in the LUS process to perform the vary on function. One of these modules is QWTLCVON, which updates the LUS peer device table to keep track of the peer device as it is allocated from the LUS process to a subsystem and back again.

- After the peer device is varied on, CPF modules are invoked in the LUS process to perform the peer LUD contact function. One of these modules is QWTLCALC, which sets up the evoke handling program (QWTLCATT) to handle any system or user evokes sent to the peer LUD. QWTLCATT receives the evoke request and examines it for validity. The LUS process is told about any system evokes and all user evoke requests are rejected.

- When a subsystem with communications work entries is started, the subsystem sends one or more requests to the LUS process to allocate a peer device. Each request is handled in the LUS process by the peer device request module (QWTLEDRQ), which sets up a peer device request in the peer device table and, if the requested peer device is available, invokes the allocate peer device to the subsystem module (QWTLCAMS). QWTLCAMS performs the allocation function and tells the subsystem of the allocation.

- When a subsystem with allocated peer devices is terminated, the peer devices are allocated to the LUS process again and the peer device allocated module (QWTLEOLK) is invoked in the LUS process. QWTLEOLK tries to allocate the peer device to another subsystem or, if no subsystem request is available, QWTLEOLK sets up the evoke handling program (QWTLCATT) to handle any system or user evoke requests that are sent to the peer device.

- If an evoke request is received at a peer device, but no conversation is available to receive the data, the evoke request maximum exceeded module (QWTLEARM) is invoked in the LUS process. QWTLEARM rejects the evoke request.

- When a peer device is varied off, CPF modules are invoked in the LUS process to perform the vary off function. One of these modules, QWTLCVOF, removes data regarding the peer device from the peer device table so that the peer device cannot be allocated by a subsystem.

Create Job Structure

Each job initiated in CPF must have an associated set of objects referred to as the job structure. These objects are divided into two groups: objects that remain after an IMPL (such as spool control block and job message queue) and temporary objects that are destroyed after an IMPL (such as process control space and program automatic storage area). The temporary part of the job structure is created by the create job structure module (QWTAMCJS). QWTAMCJS is invoked in the start CPF process during an IMPL to create a pool of the temporary job structures and is also invoked whenever the pool of temporary job structures become empty.

SNA Distribution Services

INTRODUCTION

The Distribution Services (ZD) component of CPF supports the Systems Network Architecture Distribution Services (SNADS). SNADS provides an asynchronous (delayed delivery) data distribution capability for data such as documents, messages, files, or objects. Distribution Services provides the transport mechanism so that the end user can distribute data objects to recipients that are located at the same node (same system) or other nodes that have been configured in the SNADS network. Distribution Services provides an internal interface that allows different applications above it, for example DIA, to be able to use its functions. Distribution Services transports data objects to other systems using the LU 6.2 interfaces.

Distribution Services Queues

All of the Distribution Services queues are represented by an internal object. Each entry on a SNADS queue represents a distribution. The entries reference Distribution Control Blocks that contain information about the distribution, such as the recipients, the service level, and the originator.

The three main Distribution Service queues are:

- Router Director Queue
- Local Delivery Queue
- Next System Queue

There is one Local Delivery queue for each application program (or transaction program) supported by Distribution Services and two Next System Queues for every next system that is configured in the Next System Table data base file. As distributions are received from other systems or from local application programs, they are put on the Router Director Queue. Distribution queue entries are represented by Distribution Control Blocks while they are on Distribution Services Queues. Distribution queue entries on the Router Director Queue are removed by the Distribution Services Router and routed to a Local Delivery Queue, Next System Queues, or both. A single distribution queue entry may be routed to multiple queues if it has recipients that require multiple destinations. The process of splitting up the distribution queue entry is known as fanout.

Distribution queue entries on the Next System Queues may be rerouted, held, deleted, or have their queue changed. These operations are provided by the DSPDSTSTS command. Distribution Services allocates Distribution Control Blocks as needed. When a Distribution Control Block is no longer needed to represent a distribution queue entry on a SNADS queue, it is moved to a queue that pools them so that they may be reused when needed.

GENERAL OVERVIEW

Note: An arrow identifies the module as being an entry module into the component. Indentation shows its dependency on a previous module.

Distribute Data/Status Modules

-->QZDDDTST–Distribute Data and Status: This module validates parameters passed on macros ZDDSTDTA and ZDDSTSTS, and uses the ZDLCKREL macro to lock any data objects. It also builds a Distribution Control Block and queues it on the Router Director Queue.

QZDENQRQ–Enqueue Request: This module retrieves the next free Distribution Control Block entry and moves Distribution Control Blocks from one Distribution Services Queue to another.

QZDENDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDDDSIX–Distribute Data/Status Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDDTST module.

Receive Distribution Modules

-->QZDRCVDT–Receive Distribution: This module is invoked by an application program (or transaction program), using the ZDRCVDST macro. This module retrieves the next Distribution Control Block to be received by the application program (or transaction program) from its local Delivery Queue. It builds the distribution structure used by the application program or transaction program, using the Distribution Control Block contents, and passes the distribution structure back to the invoking module.

QZDDEQRQ–Dequeue Request: This module retrieves the next Distribution Control Block to be processed on any Distribution Services queue, moves Distribution Control Blocks from any Distribution Services Queue back onto the queue of free Distribution Control Blocks, and sets the queue state of any Distribution Control Block to READY.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QSNADS Subsystem Modules

-->QZDSTRUP–Distribution Services Subsystem Startup: This module is an autostart job, started automatically when the SNADS subsystem is started by the STRSBS CL command. This module calls submitters to start the application or transaction program jobs that receive distributions from the Local Delivery Queues, and starts the Router and a Sender for every next system configured by the Next System Table data base file. Startup will also do some cleanup on the internal object representing the Distribution Services queues if necessary.

QZDSTRIE–Startup Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDSTRUP module.

Distribution Services Router

QZDROUTR–Distribution Services Router-Director: This module routes distribution queue entries, represented by Distribution Control Blocks on the Router Director Queue, to their destination Local Delivery Queues or Next System Queues. It gets the next Distribution Control Block to be routed from the Router Director Queue, determines what queues the distribution must be put on to reach its recipients and puts the distribution on those queues. It logs the distribution as being routed along with any routing found while routing it.

QZDEXTRT–Extract a Distribution Route: This module uses the Distribution Control Block distribution recipients along with the Directory, Routing Table, and Secondary Node ID Table data base files to determine which queues the Distribution Control Block should be put on, and to determine any routing time errors.

QZDDEQRQ–Dequeue Request: This module retrieves the next Distribution Control Block to be processed on any Distribution Services Queue, moves Distribution Control Blocks from any Distribution Services Queue back onto the queue of free Distribution Control Blocks, and sets the queue state of any Distribution Control Block to READY.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDENQRQ–Enqueue Request: This module retrieves the next free Distribution Control Block entry and moves Distribution Control Blocks from one Distribution Services queue to another.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDASNFB–Generate Asynchronous Feedback: This module generates a new distribution queue entry to send the errors detected while processing a distribution back to the recipient specified by the distribution.

QZDENQRQ–Enqueue Request: This module retrieves the next free Distribution Control Block entry and moves Distribution Control Blocks from one Distribution Services queue to another.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDAFBIX–Asynchronous Feedback Invocation Exit: This module does any needed cleanup on an abnormal exit of the QZDASNFB module.

QZDROUTX–Router-Director Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDROUTR modules.

Distribution Services Sender

-->QZDSNDEH–Sender Event Handler: This module is invoked when an event is signaled to a sender process due to a DSPDSTSTS operation or a distribution being routed to that sender's Next System Queue. It signals a *Start Sending* event to the QZDSENDR module. QZDSNDEH also calls QZDCKCND to create new event monitors when the entry representing the sender in the Next System Table is changed due to operations being performed.

QZDCKCND–Check Senders Conditions: This module is called to create timer event monitors for the sender's Next System Queues Start, Stop, and Force times. It checks, using the sender's Next System Table entry, if the time and/or queue depth conditions that would require the sender to start sending have been met. If any of the conditions have been met, QZDCKCND signals a *Start Sending* event to the QZDSENDR module.

-->QZDTIMEH–Sender Time Event Handler: This module is the monitor that handles sender time events created by QZDCKCND. It determines what the timer event was and calls QZDCKCND to signal a *Start Sending* event to the QZDSENDR module.

QZDCKCND–Check Senders Conditions: This module is called to create timer event monitors for the sender's Next System Queues Start, Stop, and Force times. It checks, using the sender's Next System Table entry, if the time and/or queue depth conditions that would require the sender to start sending have been met. If any of the conditions have been met, QZDCKCND signals a *Start Sending* event to the QZDSENDR module.

-->QZDSTSND–Start Sender: This module is invoked when a sender process is started either at the startup of the QSNADS subsystem or due to an Evoke request to start the sender from another SNADS node. It calls the QZDSENDR module to get the sending process going. It handles errors returned to it by QZDSENDER and if the error reported to it is recoverable it will call the QZDSENDR module again to start the sending process over again. If the error reported to it is not recoverable, it will terminate the process for this sender.

QZDSENDR–Send Distributions to the Next SNADS Node: This module is invoked to send queued distribution requests to another SNADS node. QZDSENDR calls the QZDCKCND module to determine if the current time and queue depth conditions are such that it should start sending. QZDSENDR will wait for a *Start Sending* event and upon receiving it will call QZDDEQRQ to get the next distribution queued on its Next System Queues. It calls QZDBLDRQ to build the IU form of the distribution queued and to send this IU to another SNADS node.

QZDCKCND–Check Senders Conditions: This module is called to create timer event monitors for the sender's Next System Queues Start, Stop, and Force times. It checks, using the sender's Next System Table entry, if the time and/or queue depth conditions that would require the sender to start sending have been met. If any of the conditions have been met, QZDCKCND signals a *Start Sending* event to the QZDSENDR module.

QZDBLDRQ–Build Distribution Interface Unit (IU) for the Next SNADS Node: This module uses the information in a Distribution Control Block to build the IU command and data object. It sends the IU to the receiving SNADS node and handles any Negative Acknowledge IUs received as a result of errors detected by the receiving SNADS node.

QZDDEQRQ–Dequeue Request: This module retrieves the next Distribution Control Block to be processed on any Distribution Services queue, moves Distribution Control Blocks from any Distribution Services Queue back onto the queue of free Distribution Control Blocks, and sets the queue state of any Distribution Control Block to READY.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDASNFB–Generate Asynchronous Feedback: This module generates a new distribution queue entry to send the errors detected while processing a distribution back to the recipient specified by the distribution.

QZDENQRQ–Enqueue Request: This module retrieves the next free Distribution Control Block entry and moves Distribution Control Blocks from one Distribution Services queue to another.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDAFBIX–Asynchronous Feedback Invocation Exit: This module does any needed cleanup on an abnormal exit of the AZDASNFB nodule.

QZDDEQRQ–Dequeue Request: This module retrieves the next Distribution Control Block to be processed on any Distribution Services queue, moves Distribution Control Blocks from any Distribution Services Queue back onto the queue of free Distribution Control Blocks, and sets the queue state of any Distribution Control Block to READY.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDSNDIX–Sender Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDSENDR modules.

Distribution Services LU 6.2 Receiver

-->QZDRCVR–Distribution Services LU 6.2 Receiver: This module receives data from a communications line and validates values in the IU. It calls the DIA parser to decode the IU and invokes the file server macro ZDFSWRIT to store any data objects. If the IU is received successfully it is acknowledged and logged. A Distribution Control Block is built to contain the information about the distribution received and queued on the Router Director Queue.

QZDNKSF2–Generate Negative Acknowledge IUs and Handle Suffix Type 2s: This module generates negative acknowledge IUs to report errors found when receiving an IU. The Negative Acknowledge IU is sent back to the system sending the IU. It also handles Suffix Type 2s received from a sending system that report errors found by the sending system.

QZDENQRQ–Enqueue Request: This module retrieves the next free Distribution Control Block entry and moves Distribution Control Blocks from one Distribution Services queue to another.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDRCVIX–LU 6.2 Receiver Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDRCVR modules.

Distribution Services Recovery

-->QZDRECOV–Distribution Services IPL Recovery: This module is called on an abnormal IPL or install IPL. On an abnormal IPL it will re-create the Distribution Services queues and update the Sender status in the Next System Table. On an install IPL it will create the Distribution Services queues.

Distribution Services Reclaim

-->QZDRECLM–Distribution Services Reclaim: This module is called on an RCLRSCS command. It calls file server reclaim modules for all the file servers currently supported on the system to clean up any unused file server objects using the ZDFSRECL macro.

Distribution Services General File Server Modules

QZDGFSWT–Distribution Services Write General File Server Data Object: This module generates Distribution Services file server objects. It is invoked by the Distribution Services LU 6.2 Receiver via the ?ZDFSWRIT macro when storing Distribution Services generated file server data objects. The Distribution Services General File Server is automatically invoked when an attempt is made to store a data object from a IU with a file server (as specified by the IU) that is not supported on the receiving system.

QZDGFSLR–Distribution Services Lock/Release General File Server Data Object: This module increments or decrements the SNADS usage count in Distribution Services generated file services objects. The Distribution Services General File Server is automatically invoked by Distribution Services modules via the ?ZDLCKREL macro when an attempt is made to lock or release a file server data object that was generated by using the SNADS general file server module, QZDGFSWRT. The SNADS usage count is an indication of how many times the file server data object is referenced by Distribution Control Blocks on all of the Distribution Services queues.

QZDGFSRD–Distribution Services Read General File Server Data Object: This module retrieves data from Distribution Services generated file server objects. The Distribution Services General File Server is automatically invoked by the Distribution Services Sender via the ZDFSREAD macro when an attempt is made to read a file server data object that was generated by using the SNADS general file server module, QZDGFSWRT.

QZDGFSRC–Reclaim Distribution Services General File Server Data Objects: This module will find all the Distribution Services General file server objects and delete those found that are no longer referenced by Distribution Services. This module is invoked only during the Distribution Services reclaim function via the ZDFSRECL macro to reclaim general File Server Data Objects.

Distribution Services Commands

-->QZDDSPST–Distribution Services Display Distribution Status (DSPDSTSTS) Command Processing Program: This module processes the DSPDSTSTS command. It generates the screens, and the printed output, and performs operations that allow a user to Hold, Release, and Reroute Next System Queues. It provides support for the operations that allow the Hold, Release, Remove, Reroute, and Change queues for individual distributions on the Next System Queues. It also allows the user to print the contents of the Next System Queues.

QZDENQRQ–Enqueue Request: This module retrieves the next free Distribution Control Block entry and moves Distribution Control Blocks from one Distribution Services queue to another.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDDEQRQ–Dequeue Request: This module retrieves the next Distribution Control Block to be processed on any Distribution Services queue, moves Distribution Control Blocks from any Distribution Services Queue back onto the queue of free Distribution Control Blocks, and sets the queue state of any Distribution Control Block to READY.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDASNFB–Generate Asynchronous Feedback: This module generates a new distribution queue entry to send the errors detected, while processing a distribution, back to the recipient specified by the distribution.

QZDENQRQ–Enqueue Request: This module retrieves the next free Distribution Control Block entry and moves Distribution Control Blocks from one Distribution Services queue to another.

QZDNQDQX–Enqueue Request/Dequeue Request Invocation Exit: This module does any needed cleanup on an abnormal exit of QZDDEQRQ or QZDENQRQ modules.

QZDAFBIX–Asynchronous Feedback Invocations Exit: This module does any needed cleanup on an abnormal exit of the QZDASNFB module.

QZDDSPSV–Distribution Services Display Distribution Services: (See Distribution Services DSPDSTSRV/CFGDSTSRV commands.)

QZDHELPT–Display Distribution Status Help Text: This module displays the help text for the Distribution Services command menu.

QZDDSPIX–Display Distribution Status Invocation Exit: This module does any needed cleanup on an abnormal exit of the QZDDSPST module.

Distribution Services Commands (DSPDSTSRV, CFGDSTSRV)

-->QZDDSPSV–Distribution Services Display/Configure Distribution Services Command Processing Program: This module provides the interface to the internal modules that allow the user to display, print, or configure the Routing Table, Next System Table, and the Secondary Node-ID Table.

QZDDSPNM–Display Secondary Node ID Table: This module provides the user interface that allows the user to display, print, or configure the Secondary Node-ID Table.

QZDDSPRT–Display Routing Table: This module provides the user interface that allows the user to display, print, or configure the Routing Table.

QZDDSPCF–Display Next System Table: This module provides the user interface that allows the user to display, print, or configure the Next System Table.

QZDDSPST–Distribution Services Display Distribution Status: See DSPDSTSTS command description.

QZDHELPT–Display Distribution Status Help Text: This module displays the help text for the Distribution Services command menu.

QZDDSPIX–Display Distribution Status Invocation Exit: This module does any needed cleanup on an abnormal exit of the QZDDSPST module.

Distribution Services Component Structure

Figure ZD-1 and the following text show the overall structure of the Distribution Services Component.

1 Distribute Data and Distribute Status is invoked by a locally supported application or transaction program to distribute Data Objects or Status, using the Distribution Services component.

2 Receive Distributions is invoked by a locally supported application program or transaction program to receive Data Objects or Status Distributions from the Distribution Services Component.

3 The Router Director routes distribution queue entries to the proper Distribution Services queues in the SNADS network.

4 The APPC Receiver receives distributions from other SNADS nodes in the form of Distribution Interface Units.

5 The Sender sends distributions to other SNADS nodes in the form of Distribution Units (IUs).

6 Display Distribution Status provides a user interface to display and operate on distribution queue entries on Distribution Services Next System Queues.

7 Display/Configure Distribution Services provides a user interface to allow the configuration of the SNADS network. Information about the network is kept in the Distribution Services Routing Table, Secondary Node-ID Table, and Next System Table.

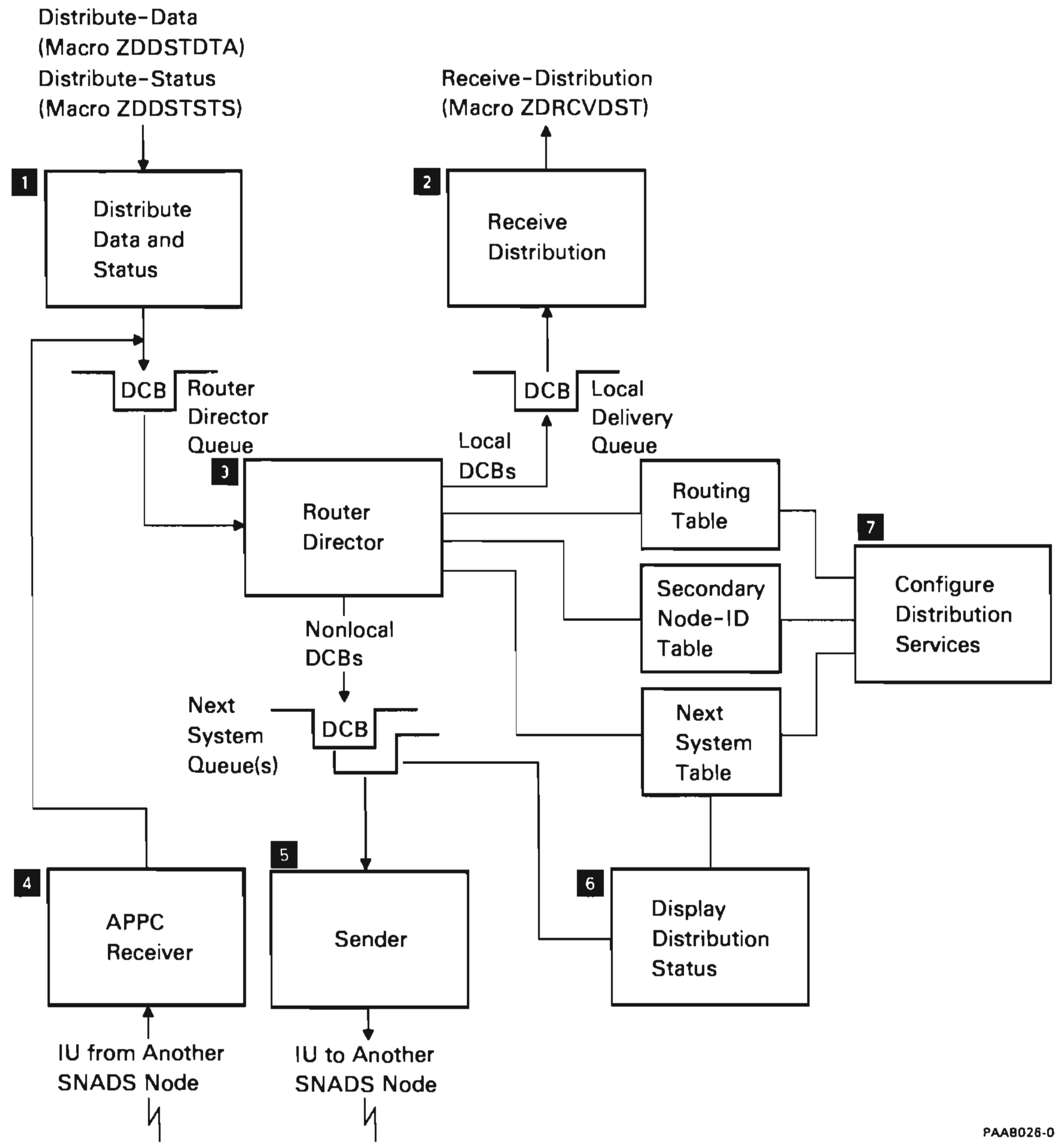


Figure ZD-1. Distribution Services Component Structure

Distribute Data and Status Module Flow

Figure ZD-2 and the following text describe how a locally supported application program or transaction program distributes data objects or status, using the Distribution Services component.

1 QZDDDTST is called by the ZDDSTDTA macro to distribute data objects or by ZDDSTSTS to distribute status. A parameter list is passed, which contains a pointer to a space containing the distribution requests description. This description includes information about the distribution requests originator, the destination application program or transaction program, the name of the data object that is being distributed and its file server, and a list of the recipients that are to receive the distribution.

2 AZDDDTST validates the values passed as part of the distribution request description including the user ID of the originator of the distribution. It validates the originator's user ID by resolving it to the user ID's recipient queue. This object must exist for every local SNADS user capable of originating a distribution.

3 If the distribution request is to distribute a data object, the data object is locked (its SNADS usage count is incremented), using the ZDLCKREL macro. The data object and file server names that were passed in the distribution request description are used as parameters to this macro invocation.

4 When all the parameters have been validated and the data object is locked using the file server specified, QZDDDTST builds a Distribution Control Block (DCB) that will contain the information about the distribution queue entry while it is queued on the Distribution Services queues. QZDDDTST calls QZDENQRQ to queue this DCB on the Router Director Queue so that it will be routed by the Router Director.

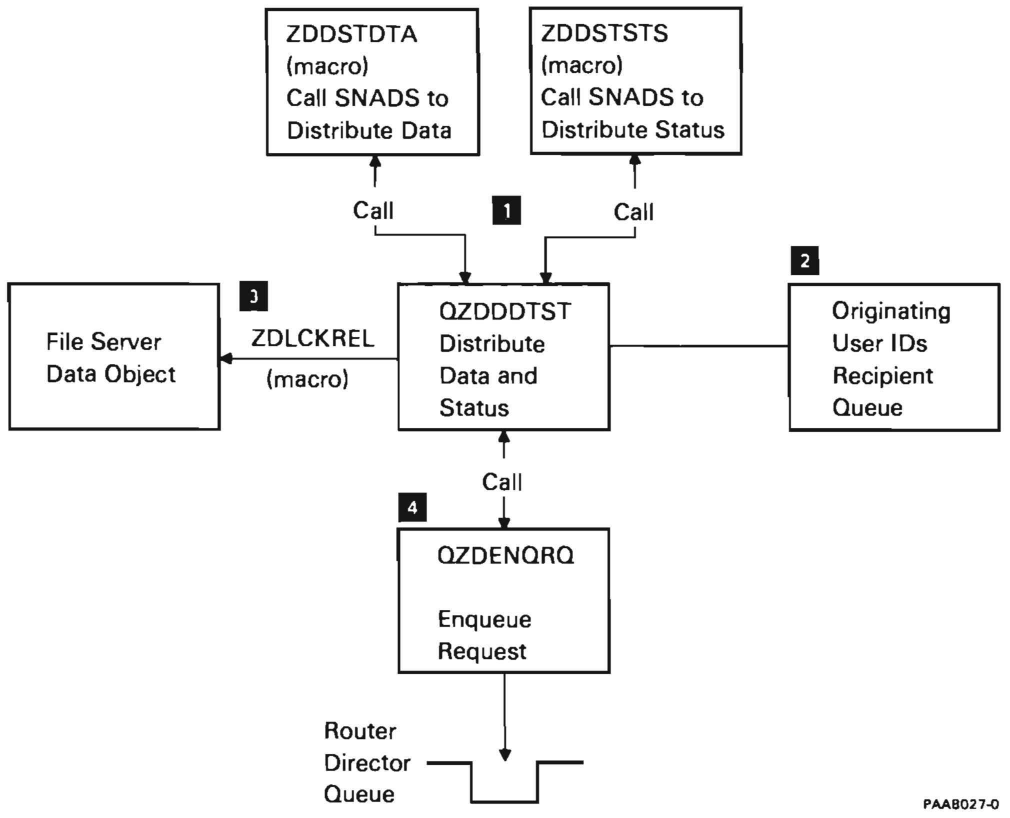

ZDDSTDTA (macro) Call SNADS to Distribute Data
ZDDSTSTS (macro) Call SNADS to Distribute Status
Call
1
Call
2
File Server Data Object
3
ZDLCKREL (macro)
QZDDDTST Distribute Data and Status
Originating User IDs Recipient Queue
Call
4
QZDENQRQ Enqueue Request
Router Director Queue
PAA8027-0

Figure ZD-2. Distribute Data and Status

Receive Distribution Module Flow

Figure ZD-3 and the following text describe how a locally supported application program or transaction program receives Data Objects or Status from the Distribution Services component.

1 QZDRCVDT is called by the ZDRCVDST macro invoked by a locally supported application program or transaction program to receive a distribution queue entry from a Distribution Services Local Delivery Queue. A parameter list, passed to the QZDRCVDT, contains the name of the Local Delivery Queue. This queue receives the entries sent by the Distribution Services Router Director.

2 QZDRCVDT calls QZDDEQRQ to retrieve the next distribution queue entry (in the form of a DCB) from the Local Delivery Queue for this application program or transaction program. Using information in the DCB removed from the Local Delivery Queue, QZDRCVDT builds and returns to the caller the information about the distribution queue entry.

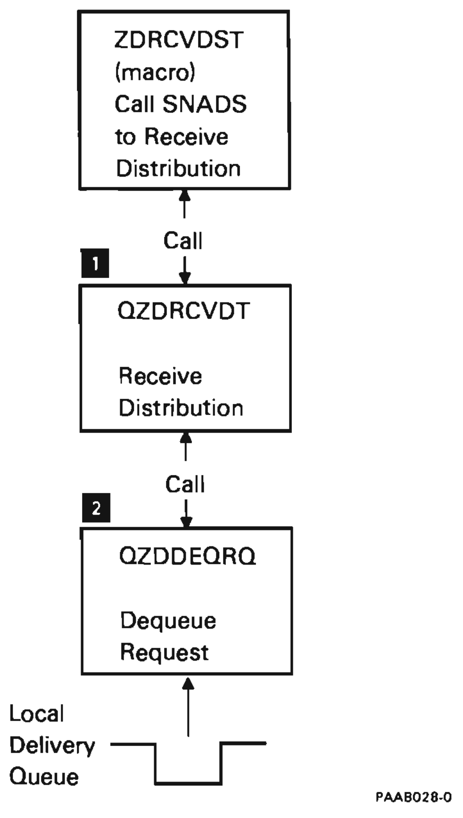

ZDRCVDST (macro) Call SNADS to Receive Distribution
Call
1
QZDRCVDT Receive Distribution
Call
2
QZDDEQRQ Dequeue Request
Local Delivery Queue
PAAB028-0

Figure ZD-3. Receive Distribution Module Flow

Router Director Module Flow

Figure ZD-4 and the following text describe how Distribution Services routes distribution queue entries to their destinations.

1 The Router Director module QZDROUTER is started when the QSNADS subsystem is started. QZDTOUTR will continue to route distribution queue entries until the QSNADS subsystem is cancelled.

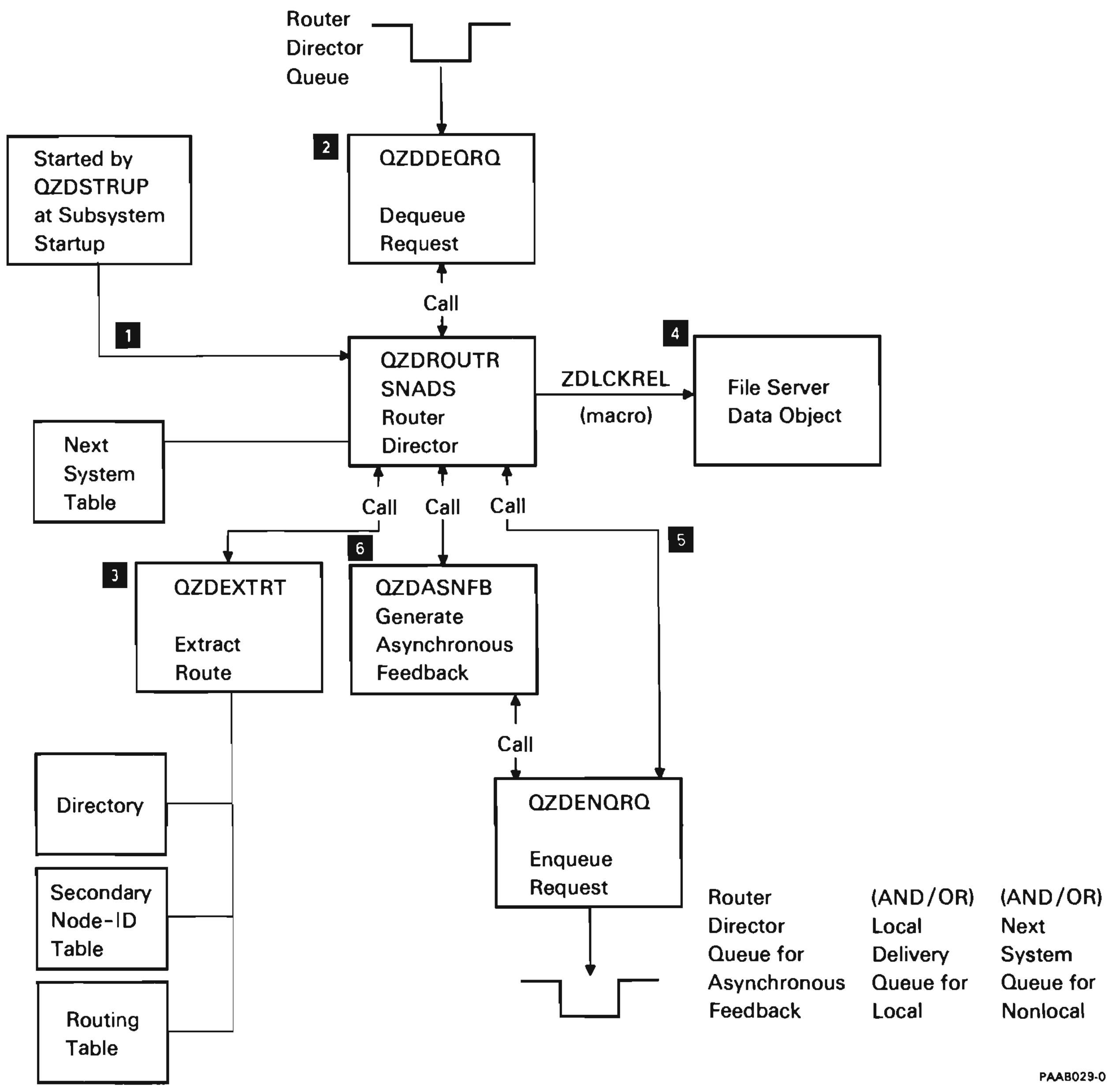
2 QZDROUTR calls QZDDEQRQ to retrieve the next distribution queue entry to be routed from the Router Director Queue. If the Router Director Queue is empty, QZDROUTR will wait until a distribution queue entry is queued on the Router Director Queue and then proceed to route it.

3 QZDROUTR then calls QZDEXTRT. Based on the list of recipients of the distribution queue entry being routed and its service level, QZDEXTRT will use information in the Routing Table, Secondary Node-ID Table, and Directory to determine which Distribution Services queue the distribution must be put on to reach its destination. It is possible that a distribution queue entry has to be routed to many destinations and therefore put on many queues. QZDEXTRT will also detect any routing errors that are the result of the distribution queue entry not being able to reach a destination as specified in the distribution queue entry list of recipients. QZDEXTRT returns this information to QZDROUTR.

4 If the distribution being routed references a file server data object, QZDROUTR invokes the ZDLCKREL macro to lock the data object. QZDROUTR will lock the data object once for every queue that it has been routed to as determined by QZDEXTRT.

5 QZDROUTR builds a Distribution Control Block (DCB) to represent the distribution on each queue that the distribution queue entry must be routed to. QZDROUTR attaches a list of the distributions recipients that required the distribution queue entry to be routed to this queue. QZDROUTR calls QZDENQRQ once for every DCB built to put it on the Local Delivery Queue or Next System Queue to which it was routed. It uses the Next System Table to obtain data to signal a *Start Sending* event to the Sender that services the Next System Queue when a DCB is queued on a Next System Queue.

6 The distribution queue entry being routed may request that an Asynchronous feedback distribution be generated to report any errors encountered during routing. If it is a distribution entry of this type and errors were found during the routing process, QZDROUTR calls QZDASNFB to generate a new Status feedback Distribution to be sent to a feedback destination specified by the distribution being routed. QZDASNFB builds a DCB to contain the information about the routing errors found. It calls QZDENQRQ to queue the DCB on the Distribution Services Router Director Queue so that it will be routed by the Router Director.



Router Director Queue
2
QZDDEQRQ Dequeue Request
Started by QZDSTRUP at Subsystem Startup
Call
1
QZDROUTR SNADS Router Director
4
ZDLCKREL (macro)
File Server Data Object
Next System Table
Call
Call
Call
6
5
3
QZDEXTRT Extract Route
QZDASNFB Generate Asynchronous Feedback
Call
Directory
QZDENQRQ Enqueue Request
Secondary Node-ID Table
Router Director Queue for Asynchronous Feedback
(AND/OR) Local Delivery Queue for Local
(AND/OR) Next System Queue for Nonlocal
Routing Table
PAAB029-0

Figure ZD-4. Router Director Module Flow

APPC Receiver Module Flow

Figure ZD-5 and the following text describe how Distribution Services receives distributions from other SNADS nodes.

1 QZDRCVR is initiated by an EVOKE request from another node in the SNADS network. It opens a communications file to the *REQUESTER device. The *REQUESTER device is identified as being the device which received the EVOKE request. QZDRCVR uses this communications file to send and receive Document Interchange Units (IUs) to another node.

2 QZDRCVR receives information about a distribution queue entry in the form of an IU by issuing a get to the communications file.

3 QZDRCVR invokes the IU parser through the OSPARSE macro to decode the incoming IU. QZDRCVR validates the presence and contents of parts of the IU.

4 If the IU contains data, the file server requested by the IU to save the data is invoked, using the ZDFSWRIT macro. If the requested file server is not supported, the SNADS General file server will automatically be used to store the data object.

5 QZDRCVR will build a Distribution Control Block that will store information about the distribution queue entry while it is queued on the Distribution Services queues. It calls QZDENQRQ to queue this DCB on the Router Director Queue so that it will be routed by the Router Director.

6 If any errors are encountered during the previous steps QZDRCVR calls QZDNKFS2 to send a Negative Acknowledge IU to report the error found to the sending node. This module is also called if QZDRCVR receives a Type 2 IU Suffix from the sending node indicating that the sending node detected an error while sending an IU. QZDRCVR does not build or queue a Distribution Control Block in either case.

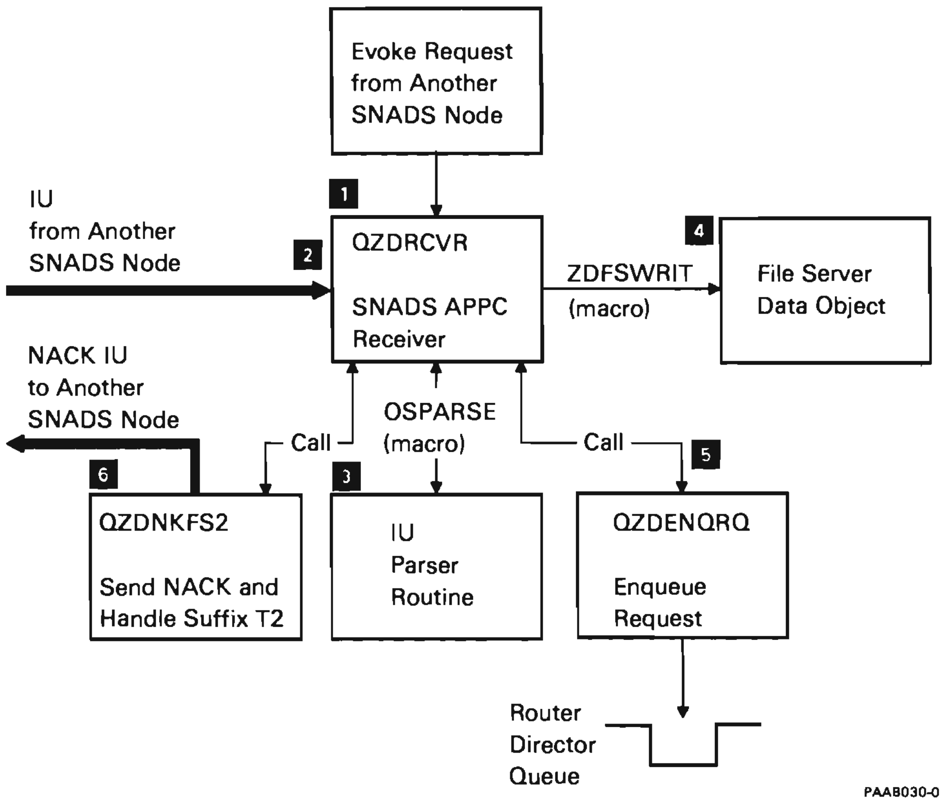


Figure ZD-5. APPC Receiver Module Flow

Sender Module Flow

Figure ZD-6 and the following text describe how Distribution Services sends distribution queue entries queued on Next System Queues to other SNADS nodes.

1 The Sender startup module QZDSTSND is started during QSNADS subsystem startup. There is a sender started in the QSNADS subsystem for every sender configured in the Next System Table. It is also possible for a sender to be started due to an EVOKE request from another SNADS node. QZDSTSND calls QZDSENDR to begin the process of sending distribution to another SNADS node. QZDSTSND handles errors reported to it from QZDSENDR, restarting the sending process if it determines the error is recoverable. If the error is not recoverable, QZDSTSND will end the sending process.

2 This module is called by QZDSTSND to begin the process of sending distribution queue entries to another SNADS node. It establishes a conversation with the APPC communications device (specified in its Next System Table entry) over which it will send distribution queue entries to that node. It then calls QZDCKCND to check the current send conditions to determine if QZDSENDR should start sending. QZDSENDR will wait for a *Start Sending* event. Upon receiving such an event, QZDSENDR will call QZDDEQRQ to retrieve the next distribution queue entry to be sent from the Next System Queue that it serves and then calls QZDBLDRQ to send the distribution queue entry to the next node. When QZDBLDRQ returns after successfully having sent the distribution queue entry, QZDSENDR will invoke the ZDLCKREL macro to unlock the data object (decrement its usage count) for the distribution queue entry that was sent.

3 QZDCKCND is called by the QZDSENDR module to check the current time and queue depth conditions against those that are specified by the sender's Next System Table entry. If QZDCKCND determines that the conditions are such that the sender should start sending distribution queue entries to another SNADS node, it signals a *Start Sending* event to the QZDSENDR module. QZDCKCND will also set up event monitors for the sender's Next System Queue Start, Stop, and Force times as configured in the sender's Next System Table entry.

4 QZDDEQRQ is called by QZDSENDR to retrieve the next distribution queue entry to be sent from the Next System Queues that it serves.

5 QZDBLDRQ is called by QZDSENDR to build the Distribution Interface Unit (IU) form of the distribution queue entry and send it, via an APPC line, to another SNADS node. QZDBLDRQ uses the information about the distribution queue entry kept in the Distribution Control Block to build the IU.

6 If the distribution references a file server data object, QZDBLDRQ invokes the ZDFSREAD macro to retrieve the data object's data. QZDBLDRQ sends this data as part of the IU.

7 QZDBLDRQ will handle a Negative Acknowledge IU (NACK) sent by the receiving SNADS node to report any errors detected by that node while receiving the IU being sent to it. QZDBLDRQ will also detect and report errors found while sending an IU to the receiving node by generating a Type 2 Suffix and sending it to that node.

8 If QZDBLDRQ detects an unrecoverable error in sending an IU for a distribution queue entry, it will not attempt to send that distribution. The distribution being sent may request that an asynchronous feedback distribution be generated to report any errors encountered during sending. If a distribution requests an asynchronous feedback, unrecoverable errors were found during the sending QZDBLDRQ then calls QZDASNFB to generate a new Status feedback Distribution to be sent to a feedback destination specified by the original distribution queue entry. QZDASNFB builds a DCB to contain the information about the routing errors found. It calls QZDENQRQ to queue the DCB on the Distribution Services Router Director Queue so that it will be routed by the Router Director. QZDBLDRQ will then invoke the ZDLCKREL macro to unlock the data object (decrement its usage count) that was being sent when the error was found.

9 QZDSNDEH is the sender event handler module that handles events sent to a sender process when a DCB is queued on one of the Next System Queues it serves. QZDSNDEH also receives events when the sender's configuration, such as Start, Stop, or Force time, is changed due to a change made to that sender's Next System Table entry. QZDSNDEH calls QZDCKCND to check the current conditions and to signal an event to QZDSENDR to start sending if those conditions are met. If the event received by QZDSNDEH indicates that a *Start Sending* operation was done using the DSPDSTSTS interface, QZDSNDEH will send a *Start Sending* event to QZDSENDR to start the sending process.

10 QZDTIMEH is the sender event handler module that handles timer events sent to a sender process. These timer event monitors were set up by QZDCKCND, using the configuration information in the sender's Next System Table entry. When a timer event is received, QZDTIMEH calls QZDCKCND to check the current conditions and to signal an event to QZDSENDR to start sending if those conditions are met.

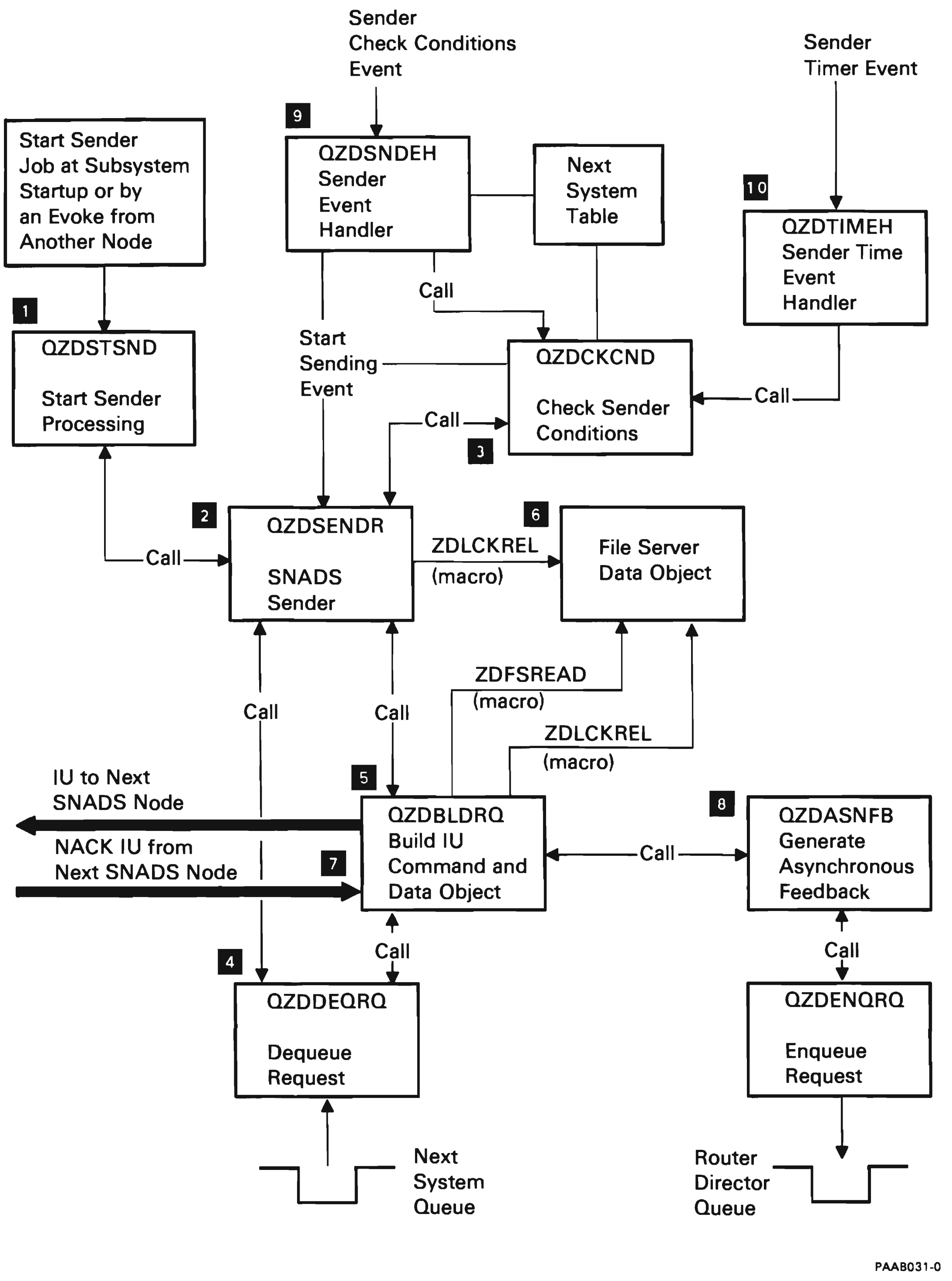

Sender Check Conditions Event
Sender Timer Event
9
QZDSNDEH Sender Event Handler
Next System Table
10
QZDTIMEH Sender Time Event Handler
Start Sender Job at Subsystem Startup or by an Evoke from Another Node
1
QZDSTSND Start Sender Processing
Call
Start Sending Event
QZDCKCND Check Sender Conditions
Call
Call
3
2
QZDSENDR SNADS Sender
6
ZDLCKREL (macro)
File Server Data Object
Call
Call
Call
ZDFSREAD (macro)
ZDLCKREL (macro)
IU to Next SNADS Node
NACK IU from Next SNADS Node
5
7
QZDBLDRQ Build IU Command and Data Object
8
QZDASNFB Generate Asynchronous Feedback
Call
Call
Call
4
QZDDEQRQ Dequeue Request
QZDENQRQ Enqueue Request
Next System Queue
Router Director Queue
PAAB031-0

Figure ZD-6. Sender Module Flow

Display Distribution Status (DSPDSTSTS) Command Module Flow

Figure ZD-7 and the following text describe the execution of the DSPDSTSTS command.

1 If the distribution queue entry removed by a remove operation references a file server data object, the ZDLCKREL macro allows the Hold, Release, Remove, Reroute, and Change of queues for individual distribution queue entries on the Next System Queues. It allows the user to Display, Hold, Release, Send, and Reroute the Next System Queues. It also provides the support that allows the user to print the contents of the Next System Queues. QZDDSPST validates the command parameters as entered.

2 When the user requests a Hold, Release, Change, or Reroute operation, an individual distribution queue entry or reroute of the queue, QZDDSPST calls QZDENQRQ to change the state of the distribution queue entry from one Distribution Services queue to another.

3 When the user requests that a distribution queue entry be removed, then QZDDSPST calls QZDDEQRQ to remove the distribution queue entry's DCB from the Distribution Services Next System Table queue it is on.

4 If the distribution queue entry removed by a remove operation references a file server data object, the ZDLCKREL macro is invoked to unlock (decrement the SNADS usage count) the file server data object referenced.

5 The distribution queue entry removed from a Next System Queue by a remove operation could be one that requests that an Asynchronous feedback distribution be generated. If it is, QZDASNFB is called to generate a new Status feedback distribution queue entry that will report its removal back to the destination specified by the distribution removed. QZDASNFB builds a DCB to contain information regarding the distribution removal. It calls QZDENQRQ to queue the DCB on the Distribution Services Router Director Queue so that it will be routed by the Router Director.

6 QZDDSPST provides an interface into the operations support that allows the user to configure Distribution Services. This is described in the overview of the DSPDSTSRV/CFGDSTSRV commands earlier in this chapter.

7 QZDHELPT provides the support for help text available to the user of the DSPDSTSTS operations.

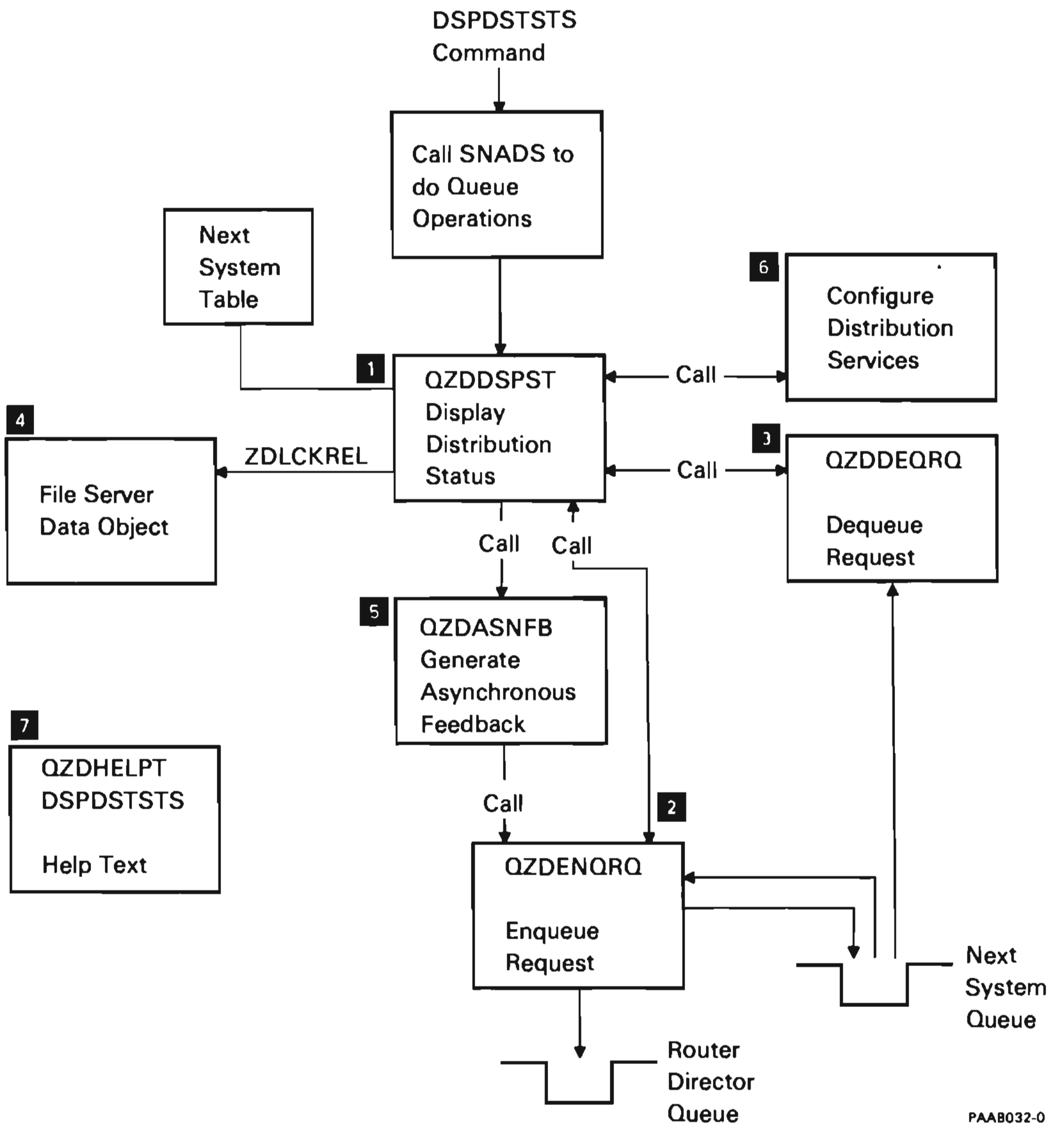

DSPDSTSTS Command
Call SNADS to do Queue Operations
Next System Table
1 QZDDSPST Display Distribution Status
6 Configure Distribution Services
Call
Call
3 QZDDEQRQ Dequeue Request
4 File Server Data Object
ZDLCKREL
Call
Call
5 QZDASNFB Generate Asynchronous Feedback
7 QZDHELPT DSPDSTSTS Help Text
Call
2 QZDENQRQ Enqueue Request
Next System Queue
Router Director Queue
PAAB032-0

Figure ZD-7. Display Distribution Status

Display/Configure Distribution Services

Figure ZD-8 and the following text describe the execution of the Display Distribution Services (DSPDSTSRV) or Configure Distribution Services (CFGDSTSRV) commands.

1 The DSPDSTSRV and CFGDSTSRV commands invoke module QZDDSTSV that provides the user interface to the Next System Table, Routing Table, and Secondary Node-ID Table. QZDDSTSV validates the command parameters as entered, displays and processes the distribution services menu if requested, and calls the appropriate module to process the selected Distribution Services Table.

2 If the user selects a menu or command option to display or configure the Routing Table, QZDDSTSV calls module QZDDSPPRT. This module provides the support for the screens and options that allow the user to display, print, or configure the Routing Table.

3 If the user selects a menu or command option to display or configure the Secondary Node-ID Table, QZDDSTSV calls module QZDDSPNM. This module provides the support for the screens and options that allow the user to display, print, or configure the Secondary Node-ID Table.

4 If the user selects a menu or command option to display or configure the Next System Table then QZDDSTSV calls module QZDDSPCF. This module provides support for the screens and options that allow the user to display, print, or configure the Next System Table.

5 QZDDSTST provides an interface with the operations support that allows the user to display and perform operations on distributions queued on the Distribution Services Next System Queues. This module can be called through the QZDDSPSV menu screen or the QZDDSPCF screens.

6 QZDHELPT provides the support for help text for the user of the DSPDSTSRV/CFGDSTSRV operations. It is called on response to a user request from any distribution services command module.

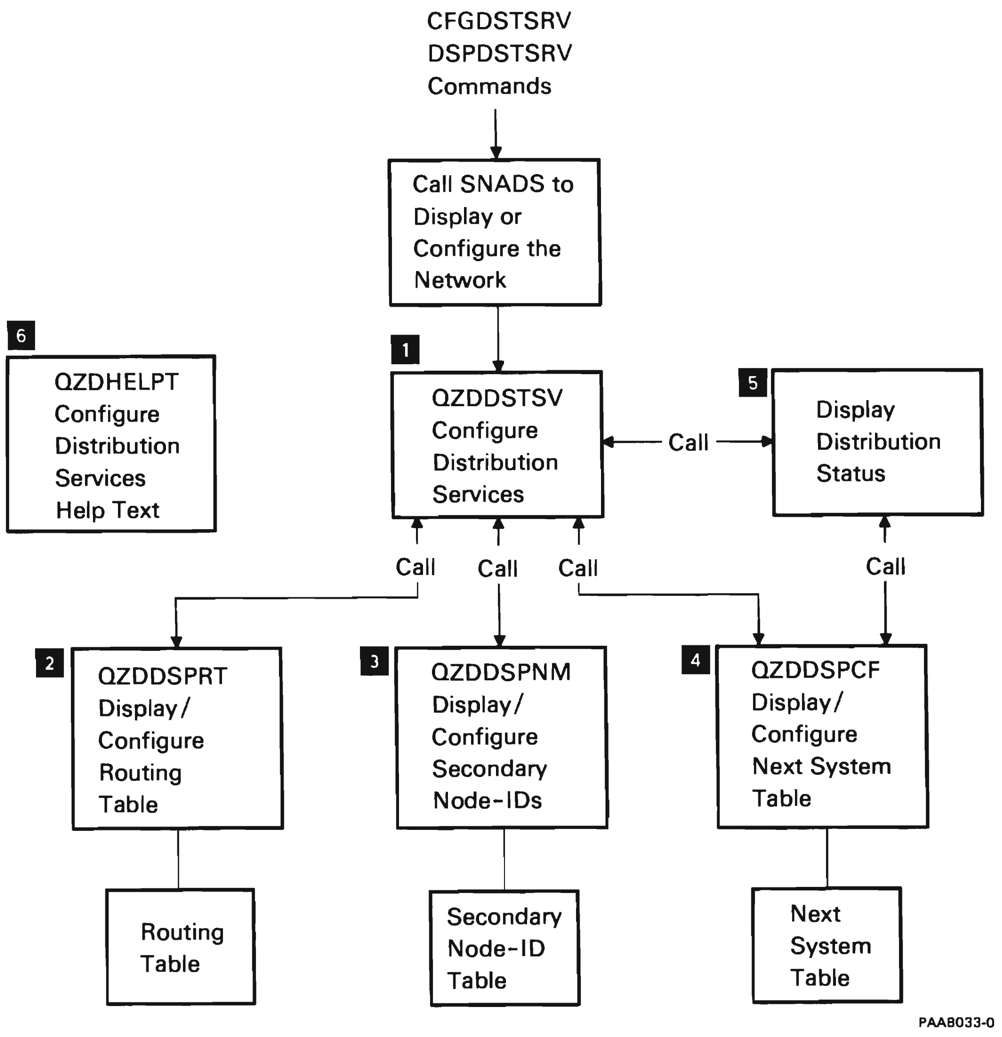

CFGDSTSRV
DSPDSTSRV
Commands
Call SNADS to Display or Configure the Network
1
QZDDSTSV Configure Distribution Services
Call
5
Display Distribution Status
6
QZDHELPT Configure Distribution Services Help Text
Call
Call
Call
Call
2
QZDDSPRT Display/ Configure Routing Table
3
QZDDSPNM Display/ Configure Secondary Node-IDs
4
QZDDSPCF Display/ Configure Next System Table
Routing Table
Secondary Node-ID Table
Next System Table
PAAB033-0

Figure ZD-8. Display/Configure Distribution Services

Appendix A. Invocation Example

INVOCATION EXAMPLE

The following is an example of the invocation of programs within a process. For each invocation there is an associated activation and invocation entry.

1 Program A is invoked as the first invocation in the process phase.

2 Program A executes a Transfer Control command to pass control to B. The invocation for A is destroyed (but not the activation) and program B is invoked and given control.

3 A Call External instruction is executed in B to invoke C.

4 An exception is signaled in C. Execution of C is suspended in order to process the exception. The external exception handler, EX1, is invoked based on the associated exception description.

5 EX1 executes a Call External command to invoke EX2.

6 EX2 executes a Return External instruction to cause its invocation to be destroyed and control to be passed to EX1 at the instruction following the Call External instruction.

7 The Return From Exception instruction is executed in EX1. This causes its invocation to be destroyed and control to be returned to C at an instruction based on the instruction address specified by the return target.

8 A Call External instruction is executed in C.

9 A Transfer Control command is executed in D. E is invoked.

10 A Call External instruction in E is executed to invoke F.

11 During the execution of F, an event occurs that is monitored within the process. Execution of F is suspended and the event handler, EV1, is invoked.

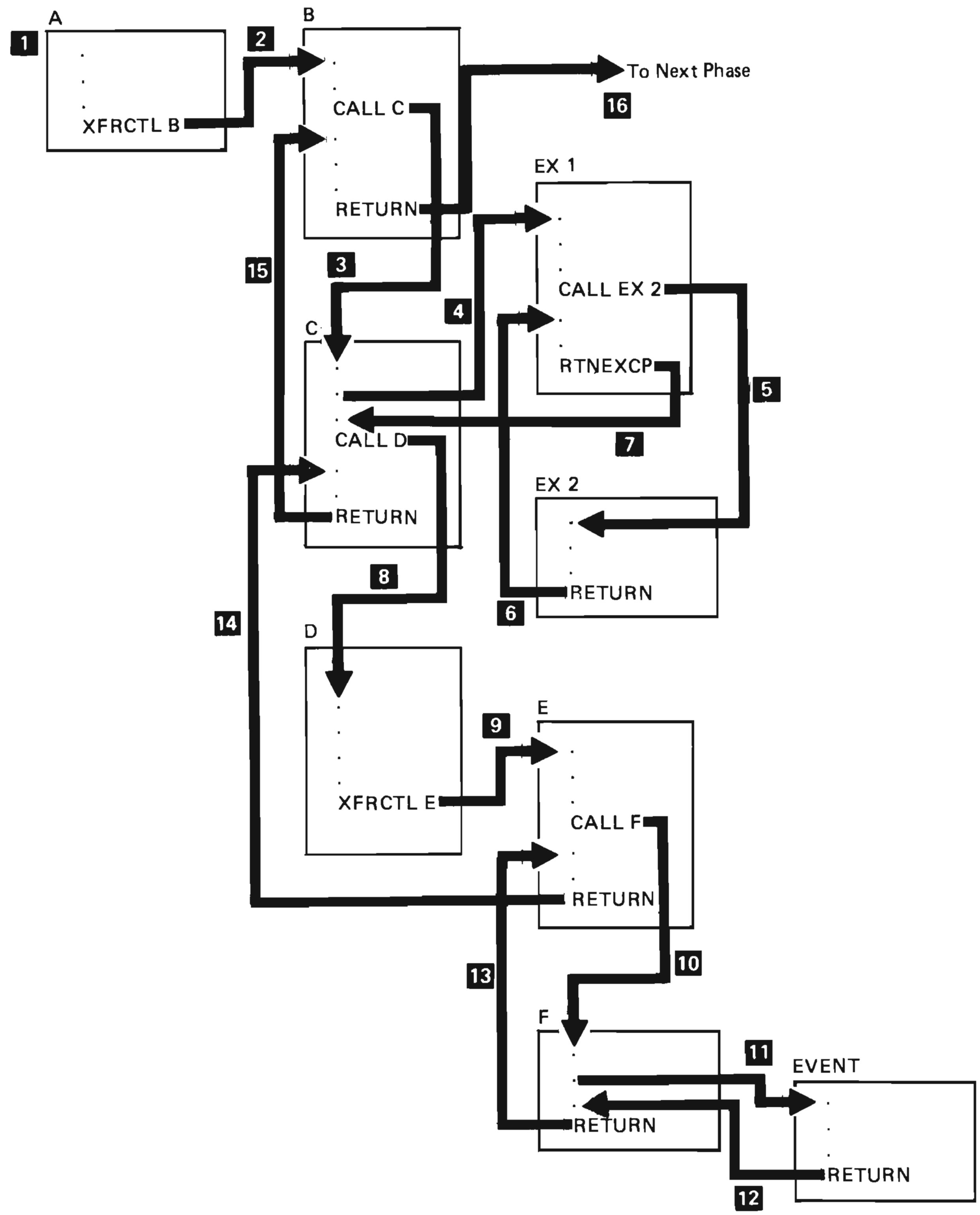
12 A Return External instruction in the event handler is executed. Invocation EV1 is destroyed and control is passed to F at the instruction following the instruction which completed execution prior to the invocation of the event handler.

13 A Return External instruction in F is executed to cause its invocation to be destroyed and control passed to the instruction following the Call External instruction in E.

14 A Return External instruction in E is executed. Control is passed to the instruction following the Call External instruction in C.

15 A Return External instruction in C is executed. The invocation of C is destroyed and control returns to the instruction following the Call External instruction in B.

16 A Return External instruction in B is executed. The invocation of B is destroyed and since it is the highest level invocation within the process, control returns to the process. If the process is in the initiation or problem phase, execution continues in the next phase. If the process is in the termination phase the process is terminated.



A
1
XFRCTL B
2
B
CALL C
RETURN
To Next Phase
16
3
15
EX 1
CALL EX 2
RTNEXCP
4
5
C
CALL D
RETURN
7
EX 2
RETURN
6
8
14
D
XFRCTL E
9
E
CALL F
RETURN
10
13
F
RETURN
11
EVENT
RETURN
12

Glossary

***CLS**: See *Class.*

***CMD**: See *command definition.*

***CUD**: See *control unit description.*

***DEVD**: See *device description.*

***DTAARA**: See *data area.*

***EDTD**: See *edit description.*

***FILE**: See *file.*

***JOBD**: See *job description.*

***JOBQ**: See *job queue.*

***LIB**: See *library.*

***LIBL**: See *library list.*

***LIND**: See *line description.*

***MSGF**: See *message file.*

***MSGQ**: See *message queue.*

***OUTQ**: See *output queue.*

***PGM**: See *program.*

***PRTIMG**: See *print image.*

***QTEMP**: See *temporary library.*

***SBSD**: See *subsystem description.*

***TBL**: See *translate table.*

***USRPRF**: See *user profile.*

abbreviated install: A process in which the object verification and damage correction part of CPF installation is done without replacing the previously installed version of CPF. Contrast with *normal install.*

access group: A system object that is a collection of other system objects, which are transferred to/from auxiliary storage as a group. The access group is used to improve storage management efficiency by specifying which system objects are used together.

access path: The means by which CPF provides a logical organization to the data in a data base file so that the data can be processed by a program. See also *arrival sequence access path* and *keyed sequence access path.*

active file: A diskette file, or tape file whose expiration date is greater than the system date.

active group job: A group job that has not been suspended by the Transfer to Group Job (TFRGRPJOB) command.

activity level: An attribute of a storage pool or the system that specifies the maximum number of jobs that can execute concurrently in the storage pool or the system.

ACTLU: An SNA command used to activate the logical unit.

ACTPU: An SNA command used to activate the physical unit.

add rights: The authority to add an entry to an object.

addressability: The ability to locate an object in online storage.

ADM: See *administrative management.*

administrative management: An IBM-supplied OFFICE/38 program that facilitates such common office tasks as the creation and maintenance of document logs, calendar, message-processing, and dictionary functions. Abbreviated ADM.

adopted user profile: The user profile that owns the program that has been created with the adopt user profile attribute. The adopted user profile supplements the process with its authorities as long as the process is executing that program. See also *propagated user profile.*

Advanced Program-to-Program Communications: Data communications support that allows a System/38 to communicate with other systems having compatible communications support. APPC is the System/38 implementation of the SNA/SDLC LU6.2 protocol. Using APPC, System/38 can start programs on another system, or another system can start programs on the System/38.

AIPL: See *alternative initial program load.*

alternative initial program load: A process, when combined with the IMPL sequence, that prepares the system for operation and installs CPF from the diskette magazine drive. Abbreviated AIPL on the operator/service panel.

APAR: See *authorized program analysis report*.

APPC: See *advanced program-to-program communication*.

application program: A program used to perform a particular data processing task such as inventory control or payroll.

argument list: A program object that provides a means of transferring object addressability from an invoking program to an invoked program. It contains a list of ODT references that specify the objects whose addressability is to be passed and the order in which the arguments are to be associated with their corresponding parameters. See also *parameter*.

arrival sequence access path: An access path that is based on the order in which records are stored in a physical file.

atomic function: An operation that, once started, must continue to completion without interruption.

attribute: Information that describes the characteristics of system objects or program objects; for example, attributes of a field include its length and data type, and attributes of a job include its user name and job date.

AUT: See *authorized user table*.

authority: The right to access objects, resources, or functions. Two kinds of authority exist, special authority and object authority.

authorization: The process of giving a user either complete or restricted access to an object, resource, or function.

authorized program analysis report: A request for correction of a problem caused by a defect in a current unaltered release of a program. A APAR.

authorized user table: A system object that contains user profile information.

automatic variable: A variable that is allocated during the invocation of the program containing the variable. Every time a program is invoked a new copy of the variable is placed in storage. Contrast with *static variable*.

autostart job: A job that is automatically initiated when a subsystem is started.

autostart job entry: A work entry in a subsystem description that specifies a job to be automatically initiated each time the subsystem is started.

auxiliary storage: All addressable storage other than main storage. Auxiliary storage is located in the system's nonremovable disk enclosures.

base pool: A storage pool containing all unassigned main storage on the system and whose minimum size is specified in the system value QBASPOOL.

batch job: A group of processing actions submitted as a predefined series of actions to be performed without a dialog between the user and the system.

batch subsystem: A subsystem in which batch jobs are to be processed. IBM supplies one batch subsystem: QBATCH.

beginning of extent: A field in a label listed in a diskette volume table of contents, which notes the beginning of a file; corresponds with end of extent.

binary synchronous communications: A form of communications line control that uses transmission control characters to control the transfer of data over a communications line. Abbreviated BSC.

BIND: An SNA command used to define the protocols for a session.

block: A set of adjacent logical records recorded as a unit on a diskette or magnetic tape.

BOE: See *beginning of extent*.

BOM: Break offset mapping.

BOMT: See *break offset mapping table*.

break delivery: When delivering messages sent to a message queue, the method that interrupts the job associated with that message queue as soon as the message arrives.

break offset mapping: See *break offset mapping table*.

break offset mapping table: A table used to validity check high-level language command names.

breakpoint: A place in a program (specified by a command or a condition) where the system halts execution and gives control to the work station user or to a specified program.

breakpoint program: For a batch job, a user program that can be invoked when a breakpoint is reached.

BSC: See *binary synchronous communications*.

BSCT: Binary synchronous communications multipoint tributary.

call: To instruct that a program is to begin execution.

called program: A program whose execution is requested by another program, a calling program.

calling program: A program that controls the execution of another program, a called program.

card device file: A device file description that contains the description of a card device file. The description identifies the device to be used and specifies the attributes of the device file.

CD: See *control unit description.*

CDO: Command definition object.

CDS: See *command definition source statement.*

CE: Customer engineer.

CF: Command function.

CI: Card image.

CL: See *control language.*

CL variable: A program variable that is declared in a control language program and is available only to that program.

class (CLS): A CPF object that specifies the execution parameters for a routing step. The class object is specified in the routing entry in a subsystem description. *CLS is the system-recognized identifier for this type of CPF object.

close: A data manipulation function that ends the connection between a file used in a program and the data or I/O device specified in the file. Contrast with *open.*

CLS: The system-recognized identifier for class, a type of CPF object. (See *class.*)

CMD: The system-recognized identifier for the definition of a command, a CPF object type. (See *command definition.*)

cold start: A process in which all noninstalled objects (CPF objects created by CPF after installation) are deleted and recreated as a group.

command: A statement used to request a function of the system. A command consists of the command name, which identifies the requested function, and parameters.

command definition: An object that contains the definition of a command (including the command name, parameter definitions, and validity checking information) and identifies the program that performs the function requested by the command. The system-recognized identifier is *CMD.

command definition source statement: A source statement used in creating a command definition. Command definition statements define key words and parameter values, qualified names, elements in a list, parameter dependencies, and prompt text for a command.

command processing program: A program that processes a command. This program performs some validity checking and executes the command so that the requested function is performed. Abbreviated CPP.

commitment control: A means of grouping file operations that allows the processing of a group of data base changes as a single unit through the COMMIT statement or the removal of a group of data base changes as a single unit through the ROLLBACK statement.

commitment definition: Information used by the system to maintain the commitment control environment throughout a routing step and, in the case of a system failure, throughout an IMPL (initial microprogram load). This information is obtained from the Begin Commitment Control command, which establishes the commitment control environment, and the file open information in a routing step.

communications file: A device file associated with a remote data processing system that describes the record formats sent to or received from that system.

completion message: A message that conveys completion status of work.

context: A system object that contains addressability to system objects by name. It is used in system pointer resolution to obtain system pointers to system objects.

control language: The set of all commands with which a user requests functions. Abbreviated CL.

control language program: An executable object that is created from source consisting entirely of control language commands.

Control Program Facility: The system support licensed program for the IBM System/38. It provides many functions that are fully integrated in the system such as work management, data base data management, job control, message handling, security, programming aids, and service. Abbreviated CPF.

control unit description: An object that contains a description of the features of a control unit that is either directly attached to the system or attached to a communications line. The system-recognized identifier is *CUD. Abbreviated CD.

controlling subsystem: An interactive subsystem that is started automatically when the system is started and through which the system operator controls the system. IBM supplies one controlling subsystem: QCTL.

conversation: The interaction between a computer and a user through a keyboard. (2) A temporary connection between an application program and an APPC session.

CPF: See *control program facility*.

CPP: See *command processing program.*

CSM: Concurrent service monitor.

CUD: See *control unit description*.

data area: An object that is used to communicate data such as CL variable values between the programs within a job and between jobs. The system-recognized identifier is *DTAARA.

data base: A structure of files and indexes that holds data and the relationship among the data. In System/38, a data base is composed of a combination of the following system objects:

- Data space–file of entries (records)
- Data space index–provides logical ordering of entries in data spaces
- Cursor–path to entries in data spaces

data base file: An organized collection of related records in the data base. See also *physical file* and *logical file*.

data communications: The transmission of data between systems and/or remote devices over a communications line.

data description specifications: A description of the user's data base or device files that is entered using a fixed-form syntax. The description is then used to create files. Abbreviated DDS.

data file: The major unit of data storage, consisting of one or more file members which contain a collection of data records stored in a user-specified format.

data management communications queue: A temporary extendable space object created outside the process access group (PAG), used to keep track of all files opened and temporarily closed during the process, file overrides that exist in the process, and common data management information for the process.

data rights: The authority to read, add, update (modify), or delete data contained in an object.

data space: A system object in which data space entries (records) are stored. Once a data space has been created, new entries can be inserted and existing entries can be updated, retrieved, or deleted.

data space entry: An ordered set of fields (record) that is contained within a data space (file). All entries within a data space have the same number of fields and identical attributes.

data space index: A system object that is used to logically order entries in one or more data spaces.

data transformation: Changing the form of data according to specific rules as data is moved between the data base and the using program. Includes changing the data type and length.

data type: An attribute used for defining data as numeric, character, or logical.

DCO: Debug communications object.

DDS: See *data description specifications*.

debug communications object: One of two principal objects used by the testing component. Unique to the program currently being debugged, the DCO contains information pertinent to the program's user-created spaces and program template components.

debug mode: The mode in which programs can be tested.

default delivery: The method of delivering messages sent to a message queue in which messages are ignored and the default reply is sent for any messages requiring a reply.

default program: A user-specified program that is assumed when no other program is specifically named on a debug command.

default record: A record in which fields are initialized with zeros if it is a numeric field or with blanks if it is a character field.

delayed maintenance: The method in which access paths are maintained that specifies that an access path is not updated when the file is closed, but updates are remembered in a *delayed* form so that they can be quickly applied at the next open, avoiding a complete rebuild. Contrast with *rebuild maintenance* and *immediate maintenance*.

delete rights: The authority to delete an entry from an object.

dequeue: An operation for removing messages from queues. Contrast with *enqueue*.

DEVD: See *device description.*

device description: An object that contains information describing a particular device that is attached to the system. The system-recognized identifier is *DEVD. Abbreviated DEVD.

device emulation: The programming that allows one device to appear to the user or to a system as another device. See also *display emulation, printer emulation, and 3270 emulation.*

device file: A file that is processed as an external input or output device attached to the system, such as a work station, a card reader/punch unit, a printer, or the diskette magazine drive. (A device file does not contain data records.)

DHCF: See *distributed host command facility.*

DIA: See *document interchange architecture.*

DIA document distribution services: The set of services that enables office users to send and receive electronic mail.

diagnostic message: A message about errors in the execution of an application program or a system function.

diskette file: A device file description that contains the description of a diskette device file. The description identifies the device to be used and specifies the attributes of the device file and of the data file on diskette.

display: A visual presentation of information on a work station screen, usually in a specific format.

display file: A device file associated with a display work station or console that describes the content of one or more displays.

display station: An input/output device containing a screen and an attached keyboard that is used as a work station to communicate with the system.

distributed host command facility: That part of a System/38 that helps to create the communications link between a System/370 terminal and a System/38 application. Abbreviated DHCF.

distribution description: A description (1 through 44 characters long) assigned to a document being distributed. It is made by the originator of the distribution and usually describes the item that is being distributed.

distribution list: A collection of system distribution directory entries. A distribution list can include users who are enrolled at any office system node. This allows users to send messages, memos, and documents to a group of users in one step.

distribution queue: In document distribution services, a list that keeps track of documents to be distributed.

distribution queue entry: In SNADS, an entry on the distribution queue indicating that an item has been passed to SNADS for distribution to one or more recipients in the SNADS network.

distribution request: See *distribution queue entry.*

distribution service level: In SNADS, the combination of priority, capacity, and protection requirements that must be satisfied to receive or route a distribution. SNADS has service levels of fast, status, data high, and data low. Items distributed with a service level of fast, status, or data high are put on the priority queue. Items distributed with a service level of data low are put on the normal queue.

distribution service unit: In SNADS, any of the nodes in a SNADS network. Abbreviated DSU.

distribution services: The support provided by CPF to receive, route, and send distributions in a SNADS network.

Document Interchange Architecture: The specification of rules and a data structure that is necessary for the predictable, coherent exchange of information between application processes. Document interchange architecture includes document library services and document distribution services. Abbreviated DIA.

document interchange session: In document interchange, the environment that allows an office system node to process requests.

document library: The system repository for filed documents and related information. Documents can be filed and retrieved by office users. On System/38, the document library is library QDOC. Contrast with *archive* and *text library.*

document library services: The set of services that enables office users to manage the contents of a document library.

document list: (1) In working with text documents, a display that lists the names of the documents contained in a particular file and allows the PS/38 user to select a document to process. (2) A logical grouping of filed documents that have common document attributes. The document list identifies which documents satisfy search criteria specified by an office user at the time the search is executed. The system-recognized identifier for the object type is *DOCL.

document name: The 1- through 44- character name of a document, assigned by the user when filing the document. Contrast with *library-assigned document name* and *document object name.*

document number: The number PS/38 assigns to a hardcopy document when a user logs that document. The first two digits of the document number are the year of logging, and the last five are in sequence, with the most recent documents having the highest number. For example, the fifth hardcopy document filed in 1985 would have the number 85-00005.

document object name: The 10-character name of a document assigned by the system when a user files the document. Contrast with *library-assigned document name* and *document name*.

DMCQ: Data management communications queue.

DS: See *data space*.

DSI: See *data space index*.

DTAARA: The system-recognized identifier for data area, a type of CPF object. (See *data area*.)

dump: To copy data in a readable format from a computer's internal storage onto an external medium such as tape, diskette, or printer.

edit: To modify a numeric field to an external format by suppressing zeros and inserting commas, periods, currency symbols, the sign status, or other constant information.

edit code: A letter or number indicating what kind of editing should be done before a field is displayed or printed.

edit description: An object that contains a description of a user-define edit code. The system-recognized identifier is *EDTD.

edit word: A word with a specific format indicating how editing should be done.

editing instructions: Instructions that are designed to allow the programmer to change the format of data items.

EDTD: The system-recognized identifier for edit description, a type of CPF object. (See *edit description*.)

element: A parameter value in a list of parameter values.

end of extent: A field in a label listed in a diskette volume table of contents, which notes the end of a file; corresponds with beginning of extent.

enqueue: An operation for placing messages on a queue. Contrast with *dequeue*.

entry job: A job that is the result of a user transferring from one subsystem to another.

entry point: A program object used to define the target instruction in an instruction stream.

entry point table: Contains pointers to functions or modules.

EOE: See *end of extent*.

EOF: End of file.

EOP: External object pointer.

EOT: End of transmission.

EPTAB: See *entry point table*.

ERAP: Error recording analysis procedure.

error log: A record of machine checks, device errors, and volume statistical data.

escape message: A message that can be monitored for and that describes a condition for which a program terminates without completing the requested function.

event: An asynchronous signal that a process can intercept.

event handler: A program, specified in an event monitor, that is to receive control when the event occurs.

event monitor: A specification of events(s) to be intercepted by a process and the event handler program to be invoked as a result.

exception: The occurrence of a monitorable machine or user-defined condition directly associated with the execution of a particular function within a program. Exceptions generally represent an abnormality detected by machine or by a program.

exception description: A program object that is used to contain information pertaining to the handling of an exception.

exclusive allow read lock state: The allocation a routing step has for an object in which other routing steps can read the object if they request a shared for read lock state for the same object. The predefined value for this lock state is *EXCLRD.

exclusive lock state: The allocation a routing step has for an object in which no other routing steps can use the object. The predefined value for this lock state is *EXCL.

external: One of the attributes of a named data program object indicating that it can be referred to by a program other than the program in which it is defined. Data pointers are used to refer to external program objects.

external entry point: Defines the first instruction to be executed in a program when it is invoked.

external message queue: A message queue that is part of the job message queue and is used to communicate between an interactive job and the work station user. For batch jobs, messages sent to the external message queue appear in the job log.

external object pointer: A pointer to an object that does not compile to inline code.

external storage: Data storage other than main or auxiliary storage.

externally described data file: A file containing data for which the fields in the records are described to CPF, by using data description specifications, when the file is created. The field descriptions can be used by the program when the file is processed. Contrast with *program-described data file*.

FCB: File control block.

FEOD: Forced end of data.

field: An area that is reserved and used for a particular item of information.

field reference file: A physical file whose record format describes the fields used by a group of files but which contains no members. The field descriptions in the field reference file can be referred to when data description specifications for other files are written. Therefore, data attributes need to be specified only once for fields used in multiple files.

file: An object that contains a description of a set of related records treated as a unit and, optionally, those records. The system-recognized identifier is *FILE.

file description: The information contained in the file that describes the file and its contents.

file overrides: The file attributes that can be specified at execution time that will override the attributes specified in the file description or in the program.

file reference function: A CPF function that lets the user track file usage on the system.

first-level message: The initial message presented to the user containing general information or designating an error.

FSM: Finite state machine.

function check: A notification (by a message) that an unexpected condition has stopped the execution of a program.

general purpose library: The library provided by CPF to contain user-oriented, IBM-provided objects and user-created objects that are not explicitly placed in a different library when they are created. Named QGPL.

generic name: A set of characters that identifies a group of objects and ends with an * (asterisk). For example, ORD*.

get operation: An I/O operation that obtains a record from an input file and passes it to a program.

GFDT: Generic function definition table.

group job: One of up to sixteen interactive jobs that are associated in a group with the same work station device and user. These jobs can be manipulated using the Change Group Attributes (CHGGRPA), Retrieve Group Attributes (RTVGRPA), Transfer to Group Job (TFRGRPJOB), and Terminate Group Job (TRMGRPJOB) commands.

group job name: A name that is assigned to an interactive job when it is changed into a group job using the Change Group Attributes (CHGGRPA) command or when a group job is started using the Transfer to Group Job (TFRGRPJOB) command. This name is used within the group to identify the job.

group job transfer: An operation performed by the Transfer to Group Job (TFRGRPJOB) command that will either start a new group job or resume an existing group job. In both cases, control is passed to the specified group job.

high-level language: A programming language that relieves the programmer from the rigors of machine level or assembler level programming; for example: RPG III and COBOL. Abbreviated HLL.

history log: A log of information about system status and events. Named QHST.

HLL: See *high-level language*.

hold delivery: The method of delivering messages sent to a message queue that holds the messages until the user asks for them.

host system: The controlling or highest level system in a data communications configuration. For example, a System/38 is the host system for the work stations connected to it.

host command facility: An IBM program product on a System/370 host system that enables a user on the System/370 to access applications on a System/38 or other systems. Abbreviated HCF.

ICO: Installation communications object.

ideographic: Pertaining to 2-byte characters consisting of pictograms, symbolic characters, and other types of symbols.

ideographic dictionary: A collection of alphameric entries with the ideographic entries that correspond to the alphameric entries.

immediate maintenance: The method by which access paths are maintained that specifies that an access path is to be updated regardless of whether the file is open. Contrast with *rebuild maintenance* and *delayed maintenance*.

impromptu message: A message created when it is sent. Contrast with *predefined message*.

information display: A display presenting information such as the status of the system to a user, but that rarely asks for a response.

informational message: A message that conveys information about the condition of a function.

initial program: A program specified in a user profile that is to be executed when the user signs on and the command processor QCL is invoked. QCL invokes the initial program.

initialize: To set to a starting position or value.

inline data file: A data file that is included as part of a job when the job is read from an input device by a reader program.

input: Information (or data) to be processed.

input field: A field in a display in which data can be entered and that is passed from the device to the program when the program reads a record.

input file: A file that contains data that is used by a program.

input stream: A group of records submitted to the system as batch input that contains CL commands for one or more jobs and/or the data records of one or more inline data files.

inquiry message: A message that conveys information but also asks for a reply.

interactive: Pertaining to a program or system that alternately accepts input and then responds. An interactive system is conversational; that is, a continuous dialog exists between the user and the system.

interactive job: A job in which the processing actions are performed in response to input provided by a work station user. During a job, a dialog exists between the user and the system.

interactive subsystem: A subsystem in which interactive jobs are to be processed. IBM supplies three interactive subsystems: QCTL, QINTER, and QPGMR.

intermediate representation of a program: The code for an object program generated by the CL compiler. The PRM (program resolution monitor) converts this code into machine interface templates, which in turn are translated into executable modules by the machine.

internal storage: All main and auxiliary storage in the system.

internal symbol table: A list formed during Phase 1 of the program resolution monitor component function, derived through lexical, syntactical, and semantic analysis of the intermediate representation of the program (IRP) source string, for use by PRM Phase 2. All program object have definitions in this table.

invocation: The execution of a program.

invocation level: Identifies the occurrence of the same program in the job's invocation stack. An invocation level is used in debug mode only. The first occurrence of a program in a job has an invocation level of 1.

invocation nesting: When more than one invocation of a specific program exists in an invocation stack.

invocation number: Identifies each program invocation in an invocation stack. The highest level program has an invocation number of 1.

invocation stack: A series of invocations.

invoke: To instruct that a specific program is to start executing.

IPL: Initial program load.

IRP: Intermediate representation of a program.

IST: See *internal symbol table*.

JMQ: Job message queue.

job: A single identifiable sequence of processing actions that represents a single use of the system. A job is the basic unit by which work is identified on the system.

job description: An object that contains the attributes of a job. The system-recognized identifier is *JOBD.

job log: A record of requests submitted to the system by a job and the messages related to them. The job log is maintained by CPF.

job message queue: A type of message queue used for receiving job requests to be processed (such as commands) and for sending messages that result from processing the job requests. A job message queue consists of a set of logical message queues that includes the external message queue. See also *external message queue* and *program message queue*.

job name: The name of a job as identified to the system. For an interactive job, the job name is the name of the work station at which the job was initiated; for a batch job, the job name is specified in the command used to submit the job.

job number: A number assigned to a job as it enters the system to distinguish the job from other jobs.

job priority: The order in which batch jobs on a job queue are to be processed. More than one job can have the same priority.

job queue: An object on which batch jobs are placed when they are submitted to the system and from which they are selected for execution by CPF. The system-recognized identifier is *JOBQ.

job queue entry: A work entry in a subsystem description that specifies the job queue from which the subsystem can accept batch jobs.

job structure queue: A temporary extendable machine interface queue that is a job structure control object. It is created during each start CPF process.

JOBD: The system-recognized identifier for job description, a type of CPF object. (See *job description*.)

JOBQ: The system-recognized identifier for job queue, a type of CPF object. (See *job queue*).

join logical file: A logical file that combines (in one record format) fields from two or more physical files. In the record format, not all the fields need to exist in all the physical files.

journal: (1) An object through which entries are placed in a journal receiver when a change is made to a data base file. The system uses the journal to record information about the journal receivers and data base files that are associated with the journal. The system-recognized identifier for the object type is *JRN. See also *journal entry* and *journal receiver*. (2) To place entries in a journal.

journal entry: A record in a journal receiver that contains information about data base files being journaled.

journal receiver: An object that contains journal entries that are generated when a change is made to a data base file being journaled. The system-recognized identifier for the object type is *JRNRCV. See also *journal*.

journaling: The process of recording changes made to a physical file member in a journal. Journaling allows the programmer to reconstruct a physical member by applying the changes in the journal to a saved version of the physical file member.

JSQ: See *job structure queue*.

Kanji: (1) The ideographic character set used by the Japanese to represent their native language. (2) A single character within that set.

key field: A field of a record whose contents are used to sequence the records of a particular type within the file.

keyed sequence access path: An access path to a data base file that is based on the contents of key fields contained in the individual records.

keyword: A name that identifies a parameter. Keywords are used in commands and in DDS.

label: (1) The name of a file on a diskette or tape. (2) An identifier of a command generally used for branching.

left-adjust: To place an entry in a field or to move the contents of a field so that the leftmost character of the data is in the leftmost position of the field.

level checking: A function that compares the record format level identifiers of a file to be opened and the file description that is part of a compiled program to determine if the file record format has changed since the program was compiled.

LIB: The system-recognized identifier for library, a type of CPF object. (See *library*.)

library: An object that serves as a directory to other objects. A library is used to group related objects and to find objects by name when they are used. The system-recognized identifier is *LIB.

library list: An ordered list of library names used to find an object. The library list indicates which libraries are to be searched and the order in which they are to be searched. The system-recognized identifier is *LIBL. *LIBL specifies to the system that a job's current library list is to be used to find the object.

LIND: See *line description*.

line description: An object that contains a description of a communications line to the system. The system-recognized identifier is *LIND. Abbreviated LIND. See also *network description*.

list element: One of several values specified in a list parameter.

local work station: A work station that is connected directly to a System/38 without need for data transmission facilities. Contrast with *remote work station.*

lock: A control applied to a system object (in behalf of a process) that guarantees the ability for a process to perform certain types of operations while prohibiting other processes from performing certain types of operations. The five types of locks are:

- LSRD–Lock for shared read
- LSRO–Lock for shared read only
- LSUP–Lock for shared update
- LEAR–Lock for exclusive use but allow read in other processes
- LENR–Lock for exclusive use with no read in other processes

lock state: A lock state defines how an object is allocated, which includes the use of the object (read or update) and whether the object can be shared (used by more than one job).

log version: A system log that contains enqueued system log messages.

logical file: A data base file through which data that is stored in one or more physical files can be accessed by means of record formats and/or access paths that are different from the physical representation of the data in the data base. Contrast with *physical file.*

logical file member: A logical grouping of data records in a logical file.

logical unit: In SNA, one of three types of network addressable units. It is a port through which an end user accesses the SNA network in order to communicate with another end user and through which the end user accesses the functions provided by the system services control point. Abbreviated LU. See also *physical unit, system services control point, primary logical unit,* and *secondary logical unit.*

logical unit description: An object that contains information describing a logical unit that is attached to the system. Abbreviated LUD.

LUD: See *logical unit description.*

machine attribute: Information pertaining to the overall system; for example, date, time of day, and machine configuration.

machine check: A type of exception that indicates a malfunction of the machine.

machine context: A system object implicitly created and maintained by the machine for maintaining addressability to certain types of system objects.

machine interface: The instruction set interface to the machine. The instruction set is called the System/38 instruction set. Abbreviated MI.

machine interface request queue: A queue used by all of the devices in a process that perform I/O. The queue is extendable, resides outside the process access group, and is addressable through the work control block.

machine pool: A storage pool used by the machine and certain highly shared CPF programs.

machine services control point: The machine component that provides services and coordinates the processing of supervisory services.

machine termination: The termination of all processes in the machine with the intent of turning power off or performing an initial microprogram load (IMPL).

main storage: The high-speed portion of machine storage used for objects and processes when they are being referred to or when they are being executed. Main storage cannot retain data while machine power is off. Contrast with *auxiliary storage.*

mapping: The reordering, conversion and selection of fields in data space entries when referred to by a program through the use of a cursor.

master debug communications object: One of two principal objects used by the testing component. Unique to the program currently being debugged, the MDCO contains information about the debug session.

MDCO: Master debug communications object.

member: An identifiable group of records that is a subset of the data base file to which it belongs. Each member conforms to the characteristics of the file and has its own access path.

menu: A type of display in which a list of options is shown.

message: A communication sent from one person or program to another person or program.

message description: The descriptive information about a message and the text of the message.

message field: An output field that is treated as a message.

message file: An object that contains message descriptions. The system-recognized identifier is *MSGF.

message identifier: A 7-character code that identifies a predefined message and is used to retrieve its message description from a message file.

message queue: An object on which messages are placed when they are sent to a person or program. The system-recognized identifier is *MSGQ.

message reference key: A key assigned to every message on a message queue, which is used to remove a message from a message queue, to receive a message, and to reply to a message.

MFCU: The abbreviation for the IBM 5424 multifunction card unit.

MI: See *machine interface.*

MICQ: Machine interface communications queue.

MIRQ: Machine interface request queue.

MPCI: Master programming change index.

MPL: See *multiprogramming level.*

MRJE: See *multi-leaving remote job entry.*

MRK: See *message reference key.*

MSGF: The system-recognized identifier for message file, a type of CPF object. (See *message file.*)

MSGQ: The system-recognized identifier for message queue, a type of CPF object. (See *message queue.*)

MTR: Machine trouble report.

multi-leaving remote job entry: The fully synchronized, two-directional transmission of a variable number of data streams between two computers using BSC facilities.

multiprogramming level: A resource management control used by CPF to provide storage pool support, by limiting the number of processes that can compete for processor cycles within the machine.

multivolume file: A file that is contained on more than one diskette or tape.

name resolution: (1) The function of resolving addressability to system objects. An unresolved system pointer specifies the symbolic name of a system object. At first reference, an unresolved system pointer is resolved as follows. The machine searches for the symbolic name in contexts until it is found and then sets addressability to the corresponding system object into the pointer, thereby making it a resolved system pointer. The contexts to be searched are contained in the name resolution list. (2) Also, the function of resolving addressability to external program objects defined in programs within a process.

name resolution list: A process attribute that is a vector of resolved system pointers to the contexts that are searched for name resolution. See also *name resolution.*

ND: See *network description.*

network description: A system object that defines and describes an I/O port and communications line for remotely attached I/O devices. The network description logically represents the I/O port to the system. The system-recognized identifier is *LIND. Abbreviated ND.

next system: In SNADS, a node in the SNADS network that is physically connected to the local system and through which distribution queue entries can be routed.

next system queue: In SNADS, a queue that is used to hold distribution queue entries that are being routed to a next system. See also *normal queue* and *priority queue.*

next system table: In SNADS, a table identifying all the next systems connected to the local system.

node: One of the systems or devices in a network.

node ID: (1) In communications, a unique string of characters that identifies a node to your system. (2) In SNADS, a two-part name by which a node is known within a SNADS network.

normal install: A process in which the CPF contained on diskettes is installed in auxiliary storage, replacing the CPF (if any) that is currently in the system. Contrast with *abbreviated install.*

notify delivery: The method of delivering messages sent to a message queue in which the work station user is notified that a message is on the queue.

notify message: A message that describes a condition for which a program requires a reply from its caller or a default reply will be sent to the program.

notify object: A message queue, a data area, or a data base file that can be used to contain information identifying the last successful commitment operation. This information can be used by the programmer to establish a restarting point for an application following an abnormal system or routing step termination. See also *commit identifier.*

NRL: See *name resolution list.*

object: A named unit that consists of a set of attributes (that describe the object) and, in some cases, data. An object is anything that exists in and occupies space in storage and on which operations can be performed. Some examples of objects are programs, files, and libraries.

object authority: The right to use a system object. There are eight object authorities:

- Object control–to control existence
- Object management–to control access and use
- Authorized pointer–to allow storing authority in a system pointer
- Space–to control access to the associated space
- Retrieve–to allow retrieving elements
- Insert–to add new elements
- Delete–to remove old elements
- Update–to modify existing elements

Contrast with *lock*.

object authorization: A specification that indicates which system objects a user can access and what rights of use have been granted relative to those system objects. See also *object authority*.

object definition table: A part of the definition of a program that defines the program objects associated with the instructions in its instruction stream. Operands of an instruction refer to entries in this table. Abbreviated ODT.

object description: The attributes (such as name, type, and owner name) that describe an object.

object existence rights: The authority to delete, save, free the storage of, restore, and transfer ownership of an object.

object information repository: An area associated with every library, containing object attributes not necessary for inclusion in the object itself. The OIR includes text, service, save/restore, special attribute, and file reference function information.

object management rights: The authority to move, rename, grant authority to, revoke authority from, and change the attributes of an object.

object owner: The user profile that owns a permanent system object. The storage occupied by the system object is charged against the owner's storage resource authorization. The owner also retains certain implied authorization rights to the system object.

object rights: The authority that controls what a system user can do to an entire object. For example, object rights can let a user delete, move, or rename an object.

object user: A user who has been authorized by the object owner to perform certain functions on an object.

OCB: Operation control block.

ODP: See *open data path*.

ODPCB: Open data path control block.

ODT: See *object definition table*.

ODT directory vector: One of the components of the object definition table (ODT). The ODV consists of a series of 4-byte entries. These entries are referred to by the operands of instructions and provide a description of the program object. The ODT entry string (OES) is used to complete the description when it cannot be completely described with the 4-byte ODV entry. Abbreviated ODV.

ODT extender string: One of the components of the object definition table (ODT). The OES contains variable-length entries specifying attributes, initial values, and other items necessary for defining program objects. OES entries are not directly referred to by System/38 instructions, but are referred to via the ODT directory vector (ODV) entries. Abbreviated OES.

ODV: See *ODT directory vector*.

OES: See *ODT extender string*.

OFCB: Output file control block.

OIR: See *object information repository*.

omit function: A CPF function that determines which records from a physical file are to be omitted from a logical file's access path.

OMT: Object mapping table.

open: The function that connects a file to a program for processing. Contrast with *close*.

open data path: The path through which all I/O operations for the file are performed. Abbreviated ODP.

operational rights: Any combination of data rights authorized to a user.

output file: A file in which information (the results of processing) is placed or displayed.

output priority: The order in which spooled output files produced by the job are to be written. More than one file can have the same priority.

output queue: An object that contains a list of output files to be written to an output device by a writer. The system-recognized identifier is *OUTQ.

PAG: See *process access group*.

parameter: Identifies an individual value or group of values to be used by a command to tailor a function requested through the command.

parameter list: A list of values that provides a means of associating addressability of data defined in a called program with data in the calling program. It contains parameter names and the order in which they are to be associated in the calling and called program.

parse: To convert a command into a format that can be used by the application or command processing programs.

PASA: See *process automatic storage area.*

password: A unique string of characters that a system user enters to identify himself to the system.

PC: See *programming change.*

PCE: Process control event.

PCO: See *process communications object.*

PCS: See *process control space.*

PCSAS: See *process control space associated space.*

PDT: See *process definition template.*

PDTPLLSP: See *process definition template/process lock list space.*

PGM: The system-recognized identifier for a program, a type of CPF object.

physical file: A data base file that contains data records. All the records have the same format; that is, they are all fixed-length records and they all contain the same fields in the same order. Contrast with *logical file.*

physical file member: A subset of the data records in a physical file.

PID: Program information department.

PLL: See *process lock list space.*

PNRL: Process name resolution list.

pointer: A special kind of data contained in a space which is distinguishable from ordinary data by the machine. A pointer can be generated only by specific machine instructions. If the contents of a pointer are altered, the machine no longer recognizes it as a pointer. There are four types of pointers:

- System pointer–addresses system objects
- Space pointer–addresses a byte location in a space
- Data pointer–describes and addresses a byte location in a space
- Instruction pointer–addresses an instruction in a program

positional list: A list of elements that further describe keyword values created by parsing a command. This list is passed to the command analyzer.

predefined message: A message whose description is created independently of when it is sent and is stored in a message file. Contrast with *impromptu message.*

primary logical unit: In SNA, the logical unit that contains the primary half-session for a particular LU-LU session. Abbreviated PLU. See also *logical unit.* Contrast with *secondary logical unit.*

print image: An object that contains a description of the print belt or train on a printer. The system-recognized identifier is *PRTIMG.

printer file: A device file description that contains the description of a printer device file. The description identifies the device to be used and specifies the attributes of the device file.

priority: The relative significance of one program to other programs. Priority specifies the relative order of resource allocation when competition for a resource is experienced.

private authority: The object authority to a system object granted to a specific user profile.

process access group: A temporary access group object that contains other temporary objects of the temporary job structure.

process automatic storage area: A space that is used for automatic program allocation when a program is invoked.

process communications object: A user convention that can be used to pass information from one process to another, outside the conventional interfaces of queues or events.

process control space: A temporary job structure in the process access group that is used by the machine as a process work area.

process control space associated space: Contains temporary job-related information and pointers to the other objects that make up the temporary job structure, also contains the job's name resolution list.

process definition template: Contains information used by CPF and the machine to initiate processes for the job. This information defines the process attributes.

process lock list space: Initialized to contain the address of the work station associated with the interactive job, the process lock list is used to transfer locks on the device (held by the monitor process) to the subprocess being initiated.

process static storage area: A space that is used for static program object allocation during program activation.

production library: A library containing objects needed for normal processing. Contrast with *test library*.

program: An object that contains a set of instructions that tell a computer where to get input, how to process it, and where to put the results. A program is created as a result of a compilation. The system-recognized identifier is *PGM.

program-described data: Data contained in a file for which the fields in the records are described in the program that processes the file. Contrast with *externally described data*.

program-described data file: A file for which the fields in the records are described only in the program that processes the file. Contrast with *externally described data file*.

program message queue: A message queue used to send messages between program invocations of a routing step.

program object: One of two MI object classifications. It includes those objects used in programs that get their definition from ODT entries. (Contrast with *system object*.

program resolution monitor (PRM): The system program that translates programs from the intermediate representation of a program (which is produced by the compiler) into program modules that can be executed under the Control Program Facility.

program variable: A named changeable value that can only exist within programs. Its value is destroyed when the program that contains it is no longer invoked.

programmer subsystem: An interactive subsystem in which programmers can perform online programming through 5251 and 5252 display stations. IBM supplies one programmer subsystem: QPGMR.

programming change: A modification to an IBM-supplied program.

programming change log: A log of information about the application of program changes and patches to IBM products. Named QCHG.

prompt: A request for information or user action. The user must respond to allow the program to proceed.

propagated user profile: An adopted user profile whose authority is propagated to other invocations. Propagation is determined by an attribute that is specified when the program is created.

protected field: A field on a display in which data cannot be keyed, changed, or erased.

PRTIMG: The system-recognized identifier for print image, a type of CPF object. (See *print image*.)

PSSA: See *process static storage area*.

public authority: The authority to an object granted to all users unless overridden by specific user authority.

put operation: An I/O operation that writes a record to an output file.

QBATCH: The IBM-supplied default batch subsystem that is used to process batch jobs.

QCE: The IBM-supplied user profile for the customer engineer (CE) who services the system hardware.

QCL: The IBM-supplied control language processor that accepts CL commands so that they can be interpreted and executed by the system.

QCTL: The IBM-supplied subsystem that can be used to process interactive jobs. (See also *interactive subsystem* and *controlling subsystem*.)

QGPL: See *general purpose library*.

QPGMR: The IBM-supplied user profile for the programmer(s) of the system.

QPSR: The IBM-supplied user profile for the programming support representative (PSR) who services the system programming.

QSECOFR: The IBM-supplied user profile for the system's security officer.

QSPL: The name of the spooling subsystems job queue and the user profile used by readers and writers. (See also *spooling subsystem*.)

QSRV: See *service library*.

QSRVLOG: See *service log*.

QSYS: See *system library.*

QSYSOPR: The IBM-supplied user profile for the system operator and the name of the system operator's message queue.

QTEMP: See *temporary library.*

qualified job name: A job name and its associated user name, or a job name and its associated user name and a system-assigned job number.

qualified object name: An object name and the name of the library containing the object.

queue: A line or list formed by items in the system waiting for service; for example, work to be performed or messages to be displayed.

quiesce: In source/sink operations, the process of completing all outstanding source/sink operations and freeing the device for a new use.

QUSER: The IBM-supplied user profile that is the default user profile for all users not having their own profiles.

read rights: The authority to read the entries in an object.

reader: A program that reads jobs from an input device and places them on a job queue.

rebuild maintenance: The method in which access paths are maintained that specifies that an access path is updated only while the file is open, not when the file is closed; the access path is rebuilt when the file is opened. Contrast with *immediate maintenance* and *delayed maintenance.*

record: An ordered set of fields that make up a single occurrence of a record format. A record is the basic unit of data transferred between a file and a program.

record format: The definition of how data is structured in the records contained in a file. The definition includes the record name and field descriptions for the fields contained in the record. The record formats used in a file are contained in the file's description.

recovery library: The library containing information related to recovery of data base operations from system failures. Named QRECOVERY.

relative record number: A number that specifies the location of a record in relation to the beginning of the file. For example, the first record in a file has a relative record number of 1.

remote device: A device whose control unit is connected to a System/38 through a data link.

remote work station: A work station whose connection to the processing system uses modems and common carrier or private data transmission facilities. Contrast with *local work station.*

reply message: A message that is a response to a received inquiry or notify message.

REQDTA: Request data save area.

request data: Data to be put in a job message queue that is used by a job. For example, a single command or group of commands.

request message: A message that requests a function from the receiving program.

response indicator: A 1-character field passed from CPF to a program with an input record to provide information about the data record or actions by the work station user.

restore: To transfer specific objects or libraries from magnetic media such as diskettes or tape to internal storage by duplicating them in internal storage. Contrast with *save.*

roll back: To remove changes that have been made to files under commitment control since the last commitment boundary.

routing data: A character string that CPF compares with character strings in the subsystem description routing entries to select the routing is to be used to initiate a routing step. Routing data can be provided by a work station user, specified in a command, or provided through the work entry for the job.

routing entry: An entry in a subsystem description that specifies the program to be invoked to control a routing step that executes in the subsystem.

routing step: The processing performed as a result of invoking a program specified in a routing entry.

RSHUTD: An SNA command used to request an orderly session shutdown.

RUTDTA: Routing data save area.

RWCB: Read write control block.

save: To transfer specific objects or libraries from internal storage to magnetic media such as diskettes or tape by duplicating them on magnetic media. Contrast with *restore.*

save system rights: The authority to save all objects. (The system operator has these rights.)

save/restore rights: The authority to save and restore all objects. (The system operator has these rights.)

SBSD: The system-recognized identifier for subsystem description, a type of CPF object. (See *subsystem description.*)

SCB: Spooling control block.

SCO: Service communications object.

SCS: Standard character set.

SDLC: Synchronous data link control.

SDT: An SNA command used to start data traffic.

second-level message: Provides additional information to that already provided in a first-level message. To obtain second-level messages, the work station user presses the Help key while a first-level message is being displayed.

secured file: A file that is protected from being overridden by an override command.

security: The control of access to or use of data or functions.

security officer: The individual at an installation who is designated to control the authorization of functions and data in the System/38.

security officer user profile: The CPF-supplied user profile that has authority to control the authorization of functions and data used in the installation. Named QSECOFR.

select function: A CPF function that determines which records from a physical file are to be selected for a logical file's access path.

select/omit field: A field in a logical file record format whose value is compared with a constant, the contents of another field, a range of values, a list of values to determine if a record is to be omitted from the access path of the logical file or selected for use by the logical file.

sequential file: A file into which records are entered one after the other. If the file is keyed, the records are processed in the sequence of the access path.

service library: The library provided in CPF that is used temporarily for loading IBM-supplied programming changes and assembling data for APAR submission. Named QSRV.

service log: The system log that contains information about the application of program changes and program patches, and the symptom strings resulting from errors. Named QSRVLOG.

session: (1) The period of time during which communication is established between the system and a user. (2) The formal bound pairing that must be established between two network addressable units before their users can communicate.

SEU: Source entry utility.

shared access path: An access path used by more than file to provide access to data common to the files. The access path specifications are contained in the description of each file that uses the access path.

shared file: A file whose ODP can be shared between two or more programs executing in the same routing step.

shared for read lock state: The allocation a routing step has for an object in which the object can be shared with another routing step if the routing step does not request exclusive use of the object. The predefined value for this lock state is *SHRRD.

shared for update lock state: The allocation a routing step has for an object in which the object can be shared either for update or for read with another routing step. The predefined value for this lock state is *SHRUPD.

shared no update lock state: The allocation a routing step has for an object in which the object can be shared with another routing step if the routing step requests either a shared no update lock state or a shared for read lock state. The predefined value for this lock state is *SHRNUP.

shared record format: A record format that is used in more than one externally described data file.

simple ODT reference: A single 2-byte operand entry that refers to a program object defined in the ODT.

SNA: See *systems network architecture*.

source: In advanced program-to-program communications, the system or program that starts jobs on another system.

source file: A file created to contain source statements for such items as high-level language programs and data description specifications.

source/sink: Devices capable of originating or accepting data signals to or from a transmission device and the data management components supporting such devices. Source/sink device include locally and remotely attached, batch and work station devices, but not the internal storage of the system.

source/sink request: The operand of a source/sink Request I/O instruction that specifies the I/O operation to be performed, the characteristics of the data to be used in the operation, and the data to be used in the operation.

source/sink resource: A system resource allocated and deallocated in units described by logical unit descriptions (device descriptions) to the process requiring the resource.

space: (1) The associated space of a system object. This is a byte-addressable region of storage that is addressed through ODT entries, space pointers, or data pointers. (2) A system object that has no functional part but is used only for its associated space.

space pointer: Contains addressability to an MI space object.

special authority: The right to perform certain system control operations, such as save/restore, save system, and job control operations.

spooled file: A device file that provides access to data processed by readers or causes output data to be saved for later processing by a writer.

spooling: The CPF-provided execution-time support that reads and writes input and output streams on an intermediate device in a format convenient for later processing or output.

spooling subsystem: A subsystem that provides the operating environment needed by the CPF programs that read jobs onto job queues and write files from the output queues. A subsystem description for a spooling subsystem is provided as part of the CPF and is named QSPL.

SPP: See *space pointer*.

SSR: See *source/sink request*.

static: One of the attributes of a program object. Objects having the static attribute are allocated space and are initiated when the program containing the program object is activated.

static variable: A variable that is allocated when a program is first invoked in a routing step and exists in storage for subsequent invocations of the same program until the program is de-activated. Contrast with *automatic variable*.

status message: A message that describes the status of the work done by a program.

storage pool: A quantity of main storage available for use by jobs executing in the storage pool. The storage pool does not consist of a given block of storage; rather it specifies an amount of storage that can be used.

subfile: A group of records of the same record format that can be displayed concurrently at a work station. The system sends the entire group of records to the work station in a single operation and receives the group in another operation.

subsystem: A predefined operating environment through which CPF coordinates work flow and resource usage.

subsystem attributes: Specifications in a subsystem description that specify the amount of main storage available to the subsystem and the number of jobs that can execute concurrently in the subsystem.

subsystem description: An object that contains the specifications that define a subsystem and that CPF uses to control the subsystem. The system-recognized identifier is *SBSD.

suspend: (1) A system object is suspended if its storage is truncated to a minimum required to maintain its existence in the machine. A suspended system object is not functionally usable until it is *loaded*. (2) A source/sink session is suspended to terminate outstanding I/O operations. (3) A process is suspended if it is made ineligible to compete for processor or main storage resources.

symbolic name: The name of a system object or an external program object. The input to a name resolution function whereby an instruction operand referring to an object by name outside its program is bound to the actual object. System pointers are resolved to system objects; data pointers are resolved to externally defined program objects.

SYP: See *system pointer*.

system arbiter: A system job that provides overall control of the work being done on the system.

system console: The keyboard and display screen on the system unit that serve as a work station for communicating with and controlling the system. See also *operator/service panel* and *work station*.

system library: The library provided by CPF to contain system-oriented objects provided as part of CPF. Named QSYS.

system log message queue: A message queue used for sending information to the system history log, service log, or change log, from any job in the system.

system object: One of two MI object classifications. It includes those MI objects whose formats are not visible above MI and all objects that are defined during execution time from attribute template operands on create instructions. These objects are referred to through system pointers. Contrast with *program object*.

system operator: The individual who operates the system and looks after the peripheral equipment necessary to initiate computer runs or finalize the computer output in the form of completed reports and documents.

system operator message queue: The message queue used by the system operator to receive and reply to messages from the system, work station users, and application programs.

system pointer: Contains addressability to an MI system object.

system termination: Putting the system in the state in which all processing is stopped. Depending on the cause of the termination, system power could be shut off (such as by a power interruption or by entering the PWRDWNSYS command) or could remain on (such as caused by a machine error condition). See also *abnormal termination* and *normal termination*.

system value: A value that contains control information for the operation of certain parts of the system. A user can change the system, or tailor the system to his working environment. System date and library list are examples of system values.

systems network architecture: The total description of the logical structure, formats, protocols, and operational sequences for transmitting information units through the communications system. Abbreviated SNA.

table: See *translate table*.

target: In advanced program-to-program communications, the program or system to which a request for processing is directed.

TBL: The system-recognized identifier for a table, a type of CPF object.

TDO: Trace data object.

template: A contiguous string of bytes that defines the attributes or values of an MI object.

temporary library: A library that is automatically created for each job to contain temporary objects that are created by that job. The library is deleted when the job ends. Named QTEMP.

temporary object: A system object that is automatically destroyed at machine termination.

termination: Putting the system or an element of the system (such as CPF or a subsystem) in the state where it no longer performs its normal function. See also *system termination*.

test library: A library to be used in debug mode and that does not contain objects needed for normal processing. Contrast with *production library*.

TIO: Trace instruction object.

token list: A list of elements, each element containing information about a part of a command. It is created by parsing a command, and is used to create a positional list.

trace: The process of recording the sequence in which the statements in a program are executed and, optionally, the values of the program variables used in the statements.

transaction: (1) In a batch or remote batch entry, a job or job step. (2) An exchange between a work station and another device that accomplishes a particular action or result; for example, the entry of a customer's deposit and the updating of the customer's balance. (3) A specific set of input data that triggers the execution of a specific processor job; a message destined for an application program. (4) A unit of processing (consisting of one or more application programs) initiated by a single request. In many cases, the request will originate at a work station. (5) For commitment control, a group of changes made to data base files that appear to the work station user to be a single change but that require multiple operations by the application program.

translate table: An object that contains a set of hexadecimal characters used to translate one or more characters of data. For example, unprintable characters can be translated to blanks, and lowercase alphabetic characters can be translated to uppercase characters. The system-recognized identifier is *TBL.

TRCO: Trace communications object.

TVO: Trace value object.

UFCB: User file control block.

UNBIND: An SNA command used to terminate a session.

update rights: The authority to change the entries in an object.

user file control block: The programmer's interface to common data management. The UFCB contains information described within the current program. Common data management uses this file and override information to create the open data path.

user identification: The ability to recognize a system user so that only the facilities and data he is authorized to use are made available to him.

user message queue: A user-created message queue used to send messages to system users and between application programs.

user name: The name by which a particular user is known to the system.

user password: A unique string of characters that a system user enters to identify himself to the system.

user profile: An object that contains a description of a particular user or group of users. A user profile contains a list of authorizations to objects and functions. The system-recognized identifier is *USRPRF.

user-defined edit code: A number (5 through 9) indicating that editing should be done on a numeric output field according to a pattern predefined to CPF. User-defined edit codes can take the place of edit words, so that repetitive coding of the same edit word is not necessary.

USRPRF: User profile.

validity checker: A user-written program that tests commands for errors in the parameter values. Validity checking is done in addition to the checking done by the command analyzer.

variable: A named modifiable value. The value can be accessed or modified by referring to the name of the variable.

view: The definition or description of a program object. See *object definition table* and *program object*.

VSSD: Virtual storage standalone dump.

VTOC: Volume table of contents.

WCB: Work control block.

WCBT: Work control block table.

WCBTE: Work control block table entry.

WCBTEQ: Work control block table entry queue.

work entry: An entry in a subsystem description that specifies a source from which jobs can be accepted to be executed in the subsystem.

work station: Elements of data processing equipment through which a system's end user has access to a computer as required for the performance of this job (work) at the physical location (station) where he performs job tasks. Examples are display/keyboard devices and printer/keyboard devices. See also *local work station* and *remote work station*.

work station controller: A device in the system unit that provides for a direct connection of local work stations to the system.

work station entry: A work entry in a subsystem description that specifies the work stations from which users can sign on to the subsystem or from which interactive jobs can transfer to the subsystem.

work station message queue: A message queue used for sending and receiving messages between work station users and between work station users and the system operator.

writer: A CPF program that writes spooled output files from an output queue to an external device, such as a printer.

X.25: In data communications, a specification of the CCITT that defines the interface to an X.25 (packet-switching) network.

X.25 feature: The feature that allows System/38 to connect to an X.25 network.

XIOM: X.25 input-output manager.

Index

IBM System/38
Control Program Facility Logic Overviews
and Component Description

LY21-0571-6

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error) check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your nearest IBM branch office to request additional publications.

Name	
Company or Organization	
Address	
	City State Zip Code
Phone No.	( )
	Area Code

No postage necessary if mailed in the U.S.A.

Cut Along Line

Fold and tape. **Please do not staple.**

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS / PERMIT NO. 40 / ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Development
Department 245
Rochester, Minnesota, U.S.A. 55901

Fold and tape. **Please do not staple.**

Cut Along Line

IBM ®

IBM®

System/38 Control Program Facility Logic Overviews and Component Description (File No. S38-36) Printed in U.S.A. LY21-0571-6

LY21-0571-6

LY21-0571-6