***Advanced Communications Function for TCAM,*** Version 3
Program Number 5665-314

IBM

# *TCAM*

# *Application Programming*
***(MVS)***

SC30-3233-1
File No. S370/4300/30XX-50

**Second Edition (June 1985)**

This edition applies to Version 3 of the Advanced Communications Function for TCAM, Program Product 5665-314. This edition also applies to all subsequent releases and modifications unless otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30XX and 4300 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

# Preface

This manual provides information on interfaces between application programs and the Advanced Communications Function for the Telecommunications Access Method. The information presented is intended to assist application programmers in coding interfaces between their programs and a TCAM message control program (MCP) in order to communicate with LUs, and other application programs. This manual does not contain any MCP programming information except that which is necessary to fully understand the interface between a TCAM MCP and a TCAM application program. *See TCAM Installation* Guide for detailed MCP information.

TCAM runs as an application program of VTAM. TCAM uses the access method services of (VTAM) Advanced Communications Functions for Virtual Telecommunications access methods and functions as an asynchronous message storing and forwarding facility. Additional processing of the message data is provided through TCAM's message handlers, which operate on the contents of the message.

TCAM application programs can be written in Basic Assembler Language (BAL), COBOL, or PL/I. It is assumed that if you use this manual, you are experienced in at least one of these languages. You should also be familiar with how to code BAL data management macros. Before using this manual, you should become familiar with the information in the *TCAM General Information* manual to the extent that you understand the fundamental concepts of TCAM.

## Contents of the Book

This manual is organized so that a programmer who has not been exposed to TCAM application programming can begin with Chapter 1; whereas, the experienced TCAM programmer can go directly to Chapter 5 and begin coding. The first four chapters of this book discuss the concepts and background necessary for coding TCAM application programs. There are very few coding details in these first four chapters.

Chapter 1 describes the TCAM environment and generally how application programs are coded. The basic concepts of passing data between TCAM and an application program are also discussed. This chapter contains no programming detail, and can be understood by anyone with a general data processing background.

Chapter 2 describes the control blocks and other TCAM supervisory information that is necessary to define the interface between TCAM and an application program. How to code, test, and execute an application program, and the starting and stopping of the TCAM interface is discussed. Specification of error exits in an application program is also discussed. This chapter contains some programming detail.

Chapter 3 describes the details of the procedure that TCAM uses to pass data to and receive data from application programs. Messages, records, and program work areas are discussed. The data transfer macros are also reviewed. Message retrieval and retransmission is discussed. This chapter contains some programming detail.

Chapter 4 describes several TCAM functions that are not directly related to the transfer of data or messages, but which are available to the application programmer. Subjects discussed include use of the operator control commands in programs, TCAM system monitoring and control, and time stamping and counting of messages. The MVS checkpoint facility and the TCAM checkpoint/restart service facility are described, and coordination of system and application checkpoints is discussed. This chapter contains some programming detail.

Chapter 5 describes the requirements and options for coding any of the TCAM application program macros. This chapter can be used as a stand-alone reference section.

Appendix A describes how to define buffers in TCAM for application programs.

Appendix B describes possible errors that can occur in coding an application program.

## Terms used in this book

Throughout this publication the term *TCAM* is used to refer to Advanced Communication Function for TCAM Version 3, unless otherwise noted. The term *network* has two meanings. A *public network* is a network established and operated by communication common carriers or telecommunication Administrations for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public. *User application network* is a configuration of data processing products, such as processors, controllers, and terminals, established and operated by users for the purpose of data processing or information exchange, which may use transport services offered by common carriers or telecommunication administrations.

*Network*, as used in this publication, refers to a user application network.

## Related Publications

In addition to being familiar with the information presented in the *TCAM General Information: Functional Description* GC30-3057 manual, you should also be familiar with the information presented in other manuals in the following list:

| Appreviated Title | Full Title | Order No. |
|---|---|---|
| *Data Management Macro Instructions* | *MVS Data Management Macros* | GC26-3873 |
| *Data Management Services* | *MVS Data Management Services Guide* | GC26-3875 |
| *JCL Reference* | *MVS JCL* | GC28-0692 |
| *Supervisor Services and Macros* | *MVS Supervisor Services and Macro Instructions* | GC28-0756 |
| *System Messages* | *MVS System Messages* | GC38-1002 |
| *TCAM General Information* | *Advanced Communications Function for TCAM, Version 3, General Information* | GC30-3057 |
| *TCAM Installation Guide* | *Advanced Communications Function for TCAM, Version 3, Installation Resource Definition, and Customization Guide* | SC30-3237 |

In addition, the following manuals may be helpful to you in writing your application program.

| Abbreviated Title | Full Title | Order No. |
|---|---|---|
| *Checkpoint/Restart* | *MVS Checkpoint Restart* | GC26-3877 |
| *TCAM Installation Reference* | *Advanced Communications Function for TCAM, Version 3, Installation Reference* | SC30-3236 |
| *TCAM Planning Guide* | *Advanced Communications Function for TCAM, Version 3, Planning Guide* | SC30-3240 |
| *TCAM Operation* | *Advanced Communications Function for TCAM, Version 3, Operation* | SC30-3239 |
| *TCAM Diagnosis Reference* | *Advanced Communications Function for TCAM, Version 3, Diagnosis Reference* | LY30-5560 |
| *TCAM Program Reference Summary* | *Advanced Communications Function for TCAM, Version 3, Program Reference Summary* | LY30-5561 |

# Summary of Changes

## SC30-3238 (June 1985)

- In order to utilize the *multiple release* facility, the user must establish two new release unique libraries. New JCL is required in order to establish these libraries. Installation procedures were changed for assembly, linkedit, and execution considerations. For examples of how to establish new libraries and JCL see "Installation Procedures, Assembly, Linkedit and Execution Considerations" on page 2-16.

# Contents

**Chapter 1. Planning TCAM Application Programs** 1-1
The TCAM Environment 1-1
The TCAM Message Control Program (MCP) 1-1
TCAM Applications 1-4
The Interface Between an MCP and an Application Program 1-7

**Chapter 2. Defining, Starting, and Stopping the Application-Program Interface** 2-1
Components of the Interface Between TCAM and an Application Program 2-1
The Control Blocks in an Application Program 2-2
The Input Data Control Block (DCB) Macro 2-3
The Output Data Control Block (DCB) Macro 2-4
Special Considerations for DCB Macro Operands 2-5
BLKSIZE 2-5
BUFL 2-5
DSORG 2-5
LRECL 2-5
MACRF 2-6
OPTCD 2-6
RECFM 2-6
STOP 2-6
SYNAD 2-7
Special Considerations for Multiple TCAMs 2-9
The Control Blocks in an MCP 2-10
PCB Macro 2-11
TPROCESS Macro 2-12
Coding TCAM Application Programs 2-13
Steps for Message Flow 2-13
Use of Data Management Macros 2-14
Secured Queues 2-15
Activating an Application Program 2-15
Deactivating an Application Program 2-15
Installation Procedures, Assembly, Linkedit and Execution Considerations 2-16
Assembling a TCAM Application Program: 2-16
Link-Editing a TCAM Application Program: 2-16
Executing a TCAM Application Program: 2-17
Testing TCAM Application Programs 2-17
Data Definition (DD) Statement Parameters 2-18
Starting and Stopping the Application Program Interface 2-19
Starting TCAM Application Programs 2-19

Opening and Closing TCAM-Related Data Control Blocks (The OPEN and CLOSE macro) 2-20
Stopping the TCAM Message Control Program (the MCPCLOSE Macro) 2-20
Summary 2-22

**Chapter 3. Transferring Data Between TCAM and an Application Program** 3-1
An Example of Message Flow Through TCAM 3-1
The TCAM Inquiry/Reply Facility 3-3
The TCAM Work Unit 3-3
Work Unit Types (Messages and Records) 3-3
Specifying Record or Message Processing 3-5
Processing a Message 3-5
Processing a Record 3-7
Work-Unit Formats 3-9
The Application Program Work Area 3-13
Defining a Static Work Area 3-13
Defining a Dynamic Work Area 3-14
Defining Optional Fields in the Work Area 3-15
The Origin or Destination Field 3-15
The Position Field 3-17
The SAM-Prefix Field 3-18
Message Handling Considerations 3-19
Specifying an End-of-Message or End-of-File Routine 3-20
Summary of Related Operands 3-22
The Data Transfer Macros 3-22
The Basic Access Technique 3-22
The READ Macro 3-23
The WRITE Macro 3-24
The CHECK Macro 3-24
Completion Codes for the Basic Access Technique 3-25
The Queued Access Technique 3-28
The GET Macro 3-28
The PUT Macro 3-28
Caution Against Issuing Simultaneous PUT and WRITE Macros 3-29
Retrieving and Retransmitting TCAM-User Messages 3-29
Retrieving Messages from TCAM Disk Data Sets (the Point Macro) 3-30
Retrieval of Multiple Messages 3-31
Summary of Related Operands 3-33
Retransmitting Messages from the TCAM Message Queue Data Set (the QRESET Macro) 3-34
Summary of Related Operands 3-36

**Chapter 4. Optional TCAM Facilities for the Application Programmer** 4-1
Issuing TCAM Operator Control Commands from an Application Program 4-1
How to Format and Issue Basic Operator Commands 4-3
Formatting Basic Operator Control Commands 4-3
Issuing Basic Operator Control Commands 4-5

Command Correlation Information Included in Extended Operator Control Replies 4-5
Initialization in the MCP for Basic Operator Control in an Application Program 4-7
Summary of Related Operands 4-8
How to Issue Extended Operator Commands from an Application Program 4-9
Direct Forwarding to Extended Operator Control 4-9
Forwarding Using End-to-End Sessions 4-9
Inspecting and Changing TCAM Control Elements 4-10
Inspecting an Entry in the Terminal Table (the TCOPY Macro) 4-11
Inspecting a Destination Queue Control Block In the MCP (the QCOPY Macro) 4-13
Changing an Entry in the Terminal Table (the TCHNG Macro) 4-14
Summary of Related Operands for the Copy and Change Macros 4-15
Inspecting and Modifying the Contents of a Fixed Header Prefix 4-15
Coordinating TCAM Checkpoints of the MCP with Checkpoints of an Application Program 4-16
The CKREQ Macro 4-17
Using the DCB Exit for Checkpoint Coordination 4-19
Coordinating MCP and Application-Program Restarts 4-20
Checkpointing Operator Control Commands 4-21
Summary of Related Operands 4-21
Determining How Many Messages are on a Specific Queue (the MCOUNT Macro) 4-22
Identifying Application-Program Input Messages by Time and Date Received (the TPDATE Macro) 4-23
Summary of Related Operands 4-23
Releasing Messages from TCAM Queues (the MRELEASE Macro) 4-23
Summary of Related Operands 4-24
Displaying Main Storage from Within an Application Program (the COREDSP Macro) 4-24
Converting Numbers into Binary, Decimal, and Hexadecimal Formats (the TCBINCNV Macro) 4-26

**Chapter 5. TCAM Application Programmer's Macro Reference Guide** 5-1
CHECK Macro 5-2
Return Codes 5-3
CKREQ Macro 5-5
Return Codes 5-6
CLOSE Macro 5-6
Return Codes 5-8
COREDSP Macro 5-8
Return Codes 5-10
DCB Macro (Input) 5-10
Return Codes 5-18
DCB Macro (Output) 5-18
Return Codes 5-24
DKJFND Macro 5-24
GET Macro 5-25
Return Codes 5-26
MCOUNT Macro 5-27
Return Codes 5-28

MCPCLOSE Macro 5-28
Return Codes 5-29
MRELEASE Macro 5-30
Return Codes 5-31
OPEN Macro 5-31
Return Codes 5-33
POINT Macro 5-33
Return Codes 5-35
PUT Macro 5-36
Return Codes 5-37
QCOPY Macro 5-38
Return Codes 5-40
QRESET Macro 5-40
Return Codes 5-42
READ Macro 5-43
Return Codes 5-45
TCBINCNV Macro 5-45
Return Codes 5-47
TCHNG Macro 5-47
Return Codes 5-49
TCOPY Macro 5-50
Return Codes 5-51
TPDATE Macro 5-52
Return Codes 5-54
WRITE Macro 5-54
Return Codes 5-56

**Appendix A. Defining Buffers in TCAM for Application Programs** A-1
Buffer Design Considerations A-1
For Application Program GET or READ Buffers A-2
For Application Program PUT or WRITE Buffers A-2
A Coding Checklist A-4

**Appendix B. Checklist of Possible Coding Errors** B-1

**Glossary** X-1
Reference Words Used in the Entries X-1

**Index** X-21

# Figures

1-1. TCAM in a Network 1-2
1-2. Examples of TCAM Message Routing and Processing 1-6
2-1. The Parameters that Define the API 2-2
2-2. Register Contents Upon Entry to the SYNAD Routine 2-8
2-3. Format of the SYNADAF Message Buffer 2-9
3-1. Work-Unit Size Determination Chart 3-8
3-2. Relative Positions of Optional Fields in the Work Area 3-10
3-3. OPTCD Coding Matrix 3-19
3-4. Completion Code Matrix 3-26
4-1. The Length of the Terminal Table Entry DSECT 4-13
4-2. Using the CKREQ Macro for Checkpoint Coordination 4-18

# Chapter 1. Planning TCAM Application Programs

This chapter describes the TCAM-VTAM environment, the types of application programs that may be coded for that environment, and the basic concepts of passing data between TCAM and an application program. This chapter contains no coding detail and can be understood by anyone with a general data processing background.

## The TCAM Environment

TCAM is a subsystem that operates as an application program of VTAM. TCAM uses the access method services of VTAM and functions as a message storing, forwarding, and queuing facility. Additional processing of the message data is provided through TCAM's message handlers (MH), which operate on the contents of the message.

Since VTAM performs the task of an access method for TCAM, details of network hardware, NCPs, controllers and links are not a direct concern of the TCAM system application programmer.

Although the application programs you write must be compatible with TCAM, they do not execute under control of TCAM. They execute under control of the same operating system that controls the host processor, TCAM and VTAM. For the remainder of this manual, TCAM application programs will be referred to simply as *programs* or *application programs.*

You can supply TCAM application programs to perform data processing functions on data which is transmitted back and forth between the application programs and other TCAM resources through the MCP.

### The TCAM Message Control Program (MCP)

The relationship between TCAM MCP and a TCAM application program is established partly by TCAM and partly by data-management in the application program. These application-program macros are discussed in detail in succeeding sections of this manual.

The main feature of the MCP with respect to an application program, is that the MCP facilitates the process of passing messages from LUs to an application program for processing, and then, if necessary, transmitting a reply to an appropriate LU. See Figure 1-1 to see how TCAM fits into the network. The application program does not need to be concerned with characteristics of the LU at which a message originated, with the transmission code of the link over which the message was sent, or with what the station control discipline had been.

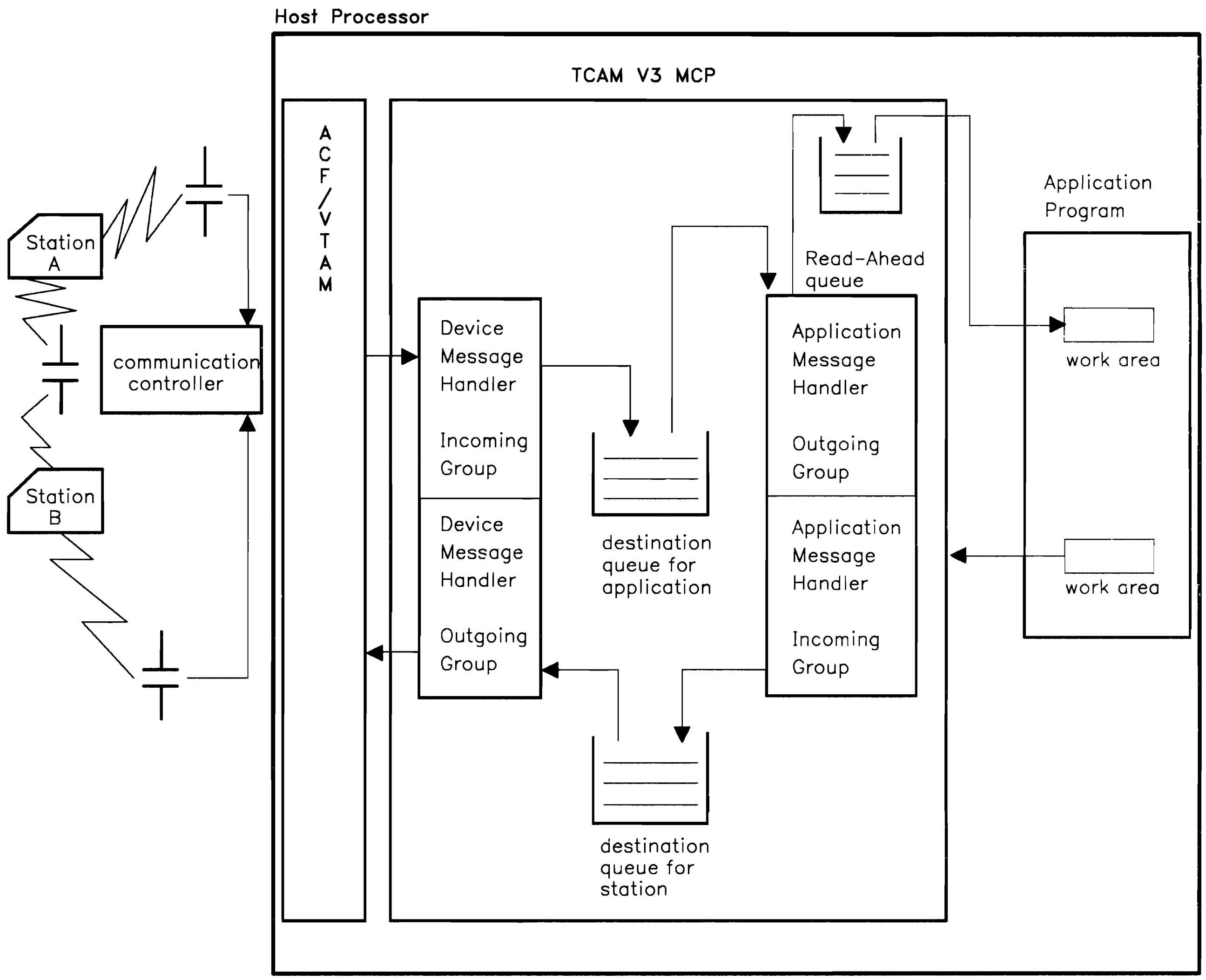


**Figure 1-1. TCAM in a Network**

An MCP with its MHs is defined by a TCAM system programmer. Your application program communicates with the network through one or more of these MHs.

The origin or destination of a message may be an external LU, an LU, or an application program. Each LU or application program defined to the MCP has an MH assigned to it. This MH handles messages flowing to or from that application program or LU.

Although an MH may handle messages for more than one LU or application program, each LU or application program has only one MH assigned to it. An MH assigned to one or more LUs is called a *device message handler* (DMH), while an MH assigned to an application program is called an *application message handler* (AMH).

If an application program can both send and receive messages, the MH assigned to it has two parts; (1) an *incoming group*, which handles messages coming into the TCAM MCP from an application program, and (2) *outgoing group*, which handles messages going out from the MCP to an application program. See "Message Flow Through a TCAM System" in Chapter 2 of *TCAM Planning Guide* for more information on the incoming group and outgoing group of MHs.

A message in the TCAM environment may be made up of header or control information, and text or user data. TCAM uses the header information to direct a message to its proper destination, and then the application program processes the text or user-data portion of the message. All messages are initially directed to the incoming group of an MH. The function of the MH is to process any control information in the message headers, and to perform the necessary TCAM functions to prepare messages for delivery to their ultimate destinations.

Once a message has entered the MCP from any source and passed through the incoming group of the MH assigned to that source, it is placed on a *destination queue*. The destination queue is a queue of messages waiting to be sent to a particular LU or application program.

To send a message to its destination, TCAM takes the message off its destination queue and routes it through the outgoing group of the MH assigned to the destination. After MH processing, the message is transmitted to the destination LU or application program.

Assume that station A in Figure 1-1 wants to direct an inquiry to the application program shown in the figure and to receive a response from the application program.

The inquiry is sent from the external LU to the MCP. It flows through the incoming group of the MH assigned to station A. The incoming group looks in a field in the message header to determine that the destination of the message is the application program. The incoming group, therefore, causes the message to be placed on the destination queue for the application program.

The message is then taken off the destination queue and flows through the outgoing group of the AMH assigned to the application program. It is then put on the Read-Ahead queue for that destination so that it will be readily available in main storage when an application program READ or GET macro is issued. When the READ or GET macro is issued by the application program, the message is transferred from the MCP buffers (read-ahead queue) to the application program work area.

When the application program issues a PUT or a WRITE macro, the reply to the inquiry is transferred from the application program's work area to the MCP buffers and is directed through the incoming group of the AMH. Since the destination of the reply is station A, the AMH incoming group causes the response to be placed on the destination queue for A.

When the time comes for the reply to be sent to station A, it is removed from the destination queue and processed by the outgoing group of the DMH assigned to station A. It is then transmitted to the external LU.

As an application programmer, you are not responsible for coding either the MCP or any of the MHs in an MCP. It will help you, however, to understand some of the details of how messages are processed before they are passed to your program. Although the following chapters occasionally refer to various functions of the MCP, the information is given only as an aid to you in coding an application program.

## TCAM Applications

TCAM application programs may be classified by how they execute. A program can execute as a batch program (batch processing), or it can process data as it is received in real time (interactive processing). How a program executes also depends on the application program environment. Regardless of how you categorize your application program, TCAM does not impose any special requirements or restrictions on the coding of that application program other than those noted in this manual.

***Batch Processing Versus Interactive Processing:*** In batch processing mode, a LU, or application program initiates a request to communicate with some other TCAM resource. The origin sends a single message or a group of messages to the MCP, and then disconnects without waiting for a reply, usually before the MCP has a chance to pass the messages on to their ultimate destination. The MCP holds complete messages that have been received error-free on a *message queue* until the destination becomes available. When the destination becomes available, it accepts the messages from the MCP and processes them. Examples of batch processing are *data collection* and *remote printing applications*.

In interactive processing mode, a connection with the MCP is established and maintained while the transmission is completed to the message destination. The message is transmitted directly through the MCP to the destination. The destination processes the message in real time, and responds to the message originator immediately. The reply is transmitted back to the originator before the communication connection is broken.

In batch mode, communication occurs most often from an external LU to an application program. Since the program may not be active while the message is being transmitted, the MCP may store or queue the message until the program can be started; alternatively, the MCP may reject the message and inform the originator that the application program is unavailable. It is also possible, however, for an application program to generate a message without first receiving input from an LU. In this case, the MCP queues the message for subsequent delivery if it cannot be delivered immediately. In interactive mode, the application program must be active and waiting in main storage before data can be transmitted to it.

The decision concerning which mode you should use depends on your assessment of the application. If it is necessary to reply to a message promptly, use the interactive mode. If the timeliness of a response is not a major consideration, the batch mode would probably be better. Whichever mode you choose, you do not need to be concerned with either transmission protocol or access methods. You should, however, thoroughly understand the application being implemented.

***Other Coding Considerations:*** Regardless of the type of processing required, there are several ways to design your application support. One generally accepted method is to code a separate program for each application; thus each application has its own interface with the MCP. This way the message will be processed by the appropriate MH, which will pass the message on to its associated application program. For instance, one program could be coded for "claims input" and a second program could be coded for "claims processing." A separate MH may be used for each program, (see Figure 1-2A.)

In some situations, however, the processing requirements may allow a single application program having a single interface with TCAM to perform all of your data processing and one MH to handle all the messages. Using this design, all messages that require application-program processing are routed to the same application program, where an analysis routine determines the type of message received, usually based upon a user-specified code in the message. The message is then transferred by this analysis routine to the appropriate processing subroutine. (See Figure 1-2B.)

Another possibility is to use an application program primarily as a message router to other application programs in the same or other address space. It performs this function by returning the messages to the MCP through PUT or WRITE macros. The application MH in the MCP then routes the message to the specified application program. (See Figure 1-2C.)

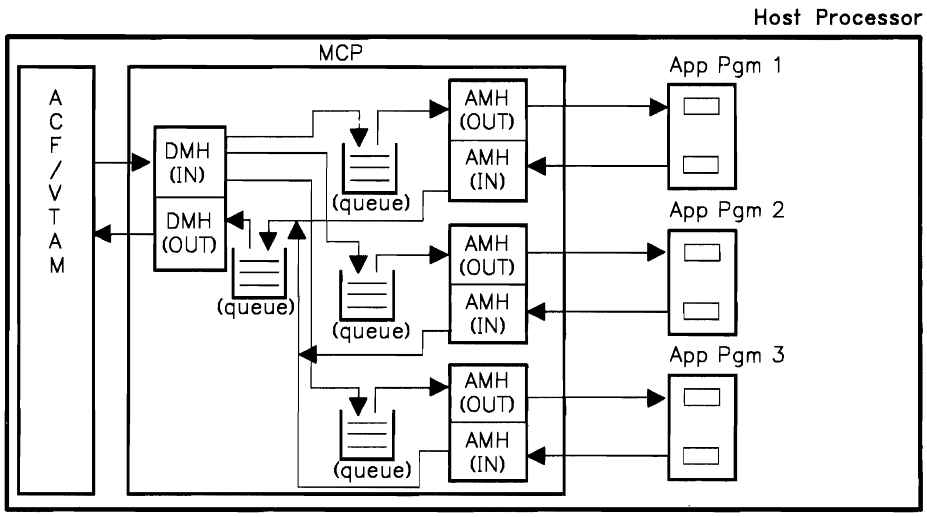


Figure 1-2A

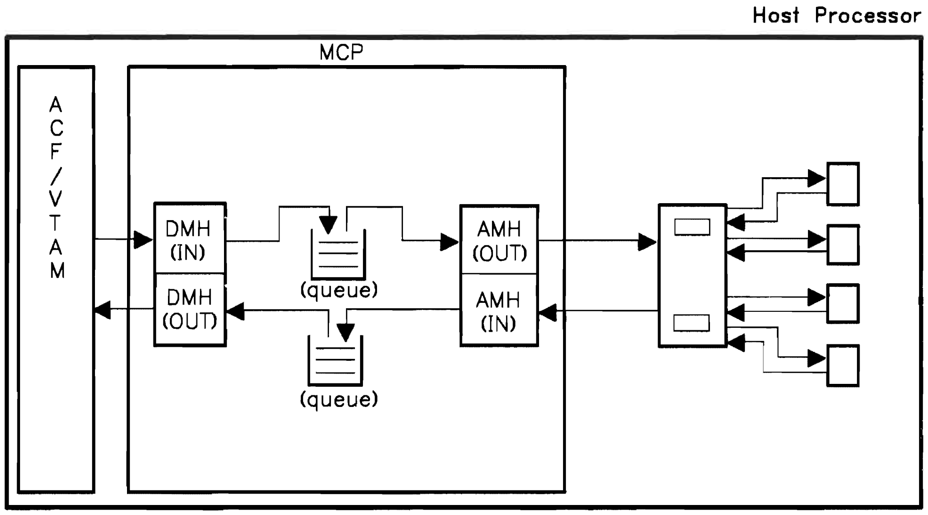


Figure 1-2B

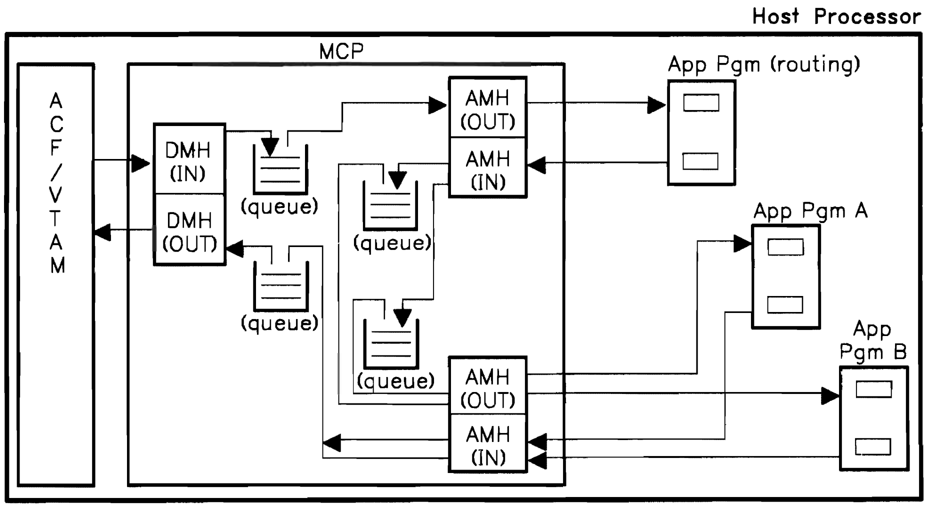


Figure 1-2C

**Figure 1-2. Examples of TCAM Message Routing and Processing**

You should discuss and coordinate examples with the TCAM system programmer and the system designer before implementing them.

For information on operating in a multiple TCAM environment, see “Special Considerations for Multiple TCAMS” in Chapter 2.

In summary, the TCAM MCP monitors and controls the transmission of messages between sources and destinations. This processing or disposition of messages at the host may be handled either by TCAM or by a “front end” application program.

## The Interface Between an MCP and an Application Program

Because an application program depends on the MCP to perform its input/output operations, you must establish a logical relationship between the application program and the MCP. This relationship is called the *application-program interface* (API).

In passing data between the MCP and an application program, there must be a common format. This format is specified through several TCAM macros. The macros allow you to specify parameters to TCAM that define what the application program expects as input and what it will provide for output. The rationale for these macros is discussed in Chapter 2, and the actual coding details are discussed in Chapter 5.

Application programs always run asynchronously with TCAM and, in most cases, in another address space, but always as a separate task or subtask. Application programs can run in the same address space as the MCP by running as attached subtasks of the TCAM initiator, or they may operate as subtasks of the MCP. Also, unless a program is running as a subtask of the initiator, any communication from the application program to a TCAM resource must involve two address spaces in the host processor. In order to communicate between regions, a format for interaction must be set up. This manual describes this interface from the standpoint of the application program. The two types of data which pass through the API are control information and message data. The control information is standard and is discussed in Chapter 2; the message data depends on the application and is discussed in Chapters 3 and 4. Where the TCAM side of the interface is discussed in this manual, only that information necessary to understand the application program side of the interface is given. Coding of the TCAM side of the API is the responsibility of the TCAM system programmer.

Because TCAM has message queuing capability, the MCP processes messages and directs them to specific destination queues, rather than immediately passing them directly to their destinations. Messages addressed to application programs will eventually be sent to a work area in the program. Conversely, application programs may be concurrently presenting data to TCAM from work areas.

There are several parameters that the application programmer and the TCAM system programmer must agree on in order to establish a compatible format for passing messages. You must establish these parameters *before* application programming begins. All of these parameters are discussed in detail in Chapters 2 through 4.

# Chapter 2. Defining, Starting, and Stopping the Application-Program Interface

This chapter describes the control blocks and other supervisory information necessary to define the interface between TCAM and an application program; how to code, test, and execute application programs; and how to start and stop the application program interface. This chapter contains programming detail that may not be understood by someone without a programming background.

## Components of the Interface Between TCAM and an Application Program

The relationship between a TCAM MCP and an application program is defined and controlled at the application program interface (API). This is a logical relationship rather than a physical one and is defined through the specification of TCAM macro operands. This relationship is necessary so that both the MCP and an application program will know the data format. The primary components of the API are two types of control blocks. One type resides in the application program and is called a data control block or (DCB); the other type resides in a TCAM MCP and is called a process control block or (PCB).

Every time your application program refers to some entity outside the program's address space, it must also refer to a related data control block. Conversely, every time the TCAM MCP communicates with your application program, it must specify a particular process control block in the MCP area of addressability. The following discussion concerns the specification of the parameters in these control blocks that are necessary to establish the API. Some of these parameters are specified in the application program and others are specified in the MCP. The choice of parameters must be coordinated so that each side of the interface will function properly. These control blocks are created by expansion of macro instructions. The application programmer is responsible for coding the DCB macro, and the TCAM system programmer is responsible for coding the PCB macro in the MCP.

To complete the interface, the system programmer must also code at least one TPROCESS macro in the MCP. A TPROCESS macro relates a single TCAM input or output message queue to your application program. These queues are referred to as process queues regardless of whether they are input or output.

Figure 2-1 depicts the specific parameters required to establish an interface, and the relationship between those parameters. The remainder of this chapter discusses how to specify those parameters and how to activate the API. The actual passing of data through the interface is discussed in Chapter 3.

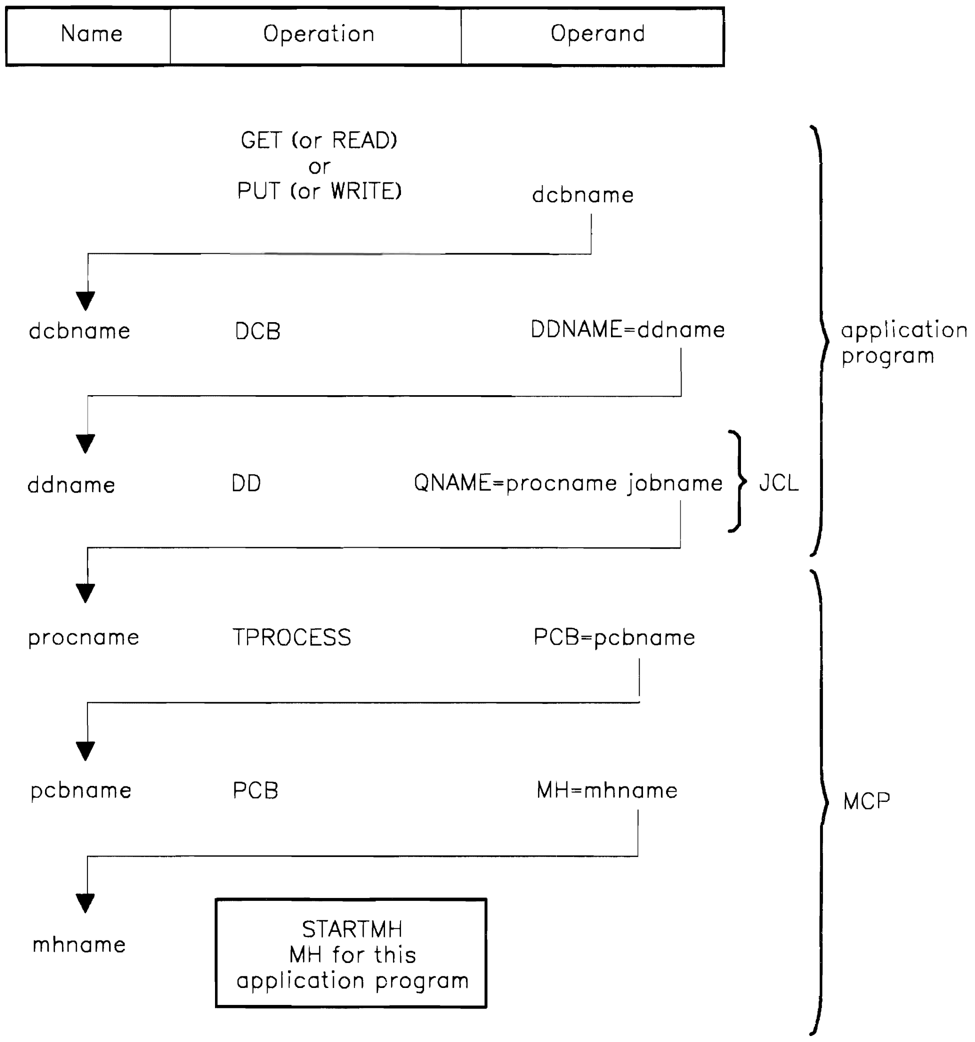


**Figure 2-1. The Parameters that Define the API**

## The Control Blocks in an Application Program

Any attempt to send or receive data in an application program must be associated with a data control block (DCB). One input DCB macro must be coded in an application program for each source of input for this specific application program. In the MCP, a TPROCESS macro creates an entry in a TCAM terminal table for a single destination queue. (For the purpose of creating queues, TCAM considers external LUs and application programs as destinations requiring a destination queue). This queue is a logical input data set for the application program.

In addition, one output DCB macro may be coded in the application program for each TPROCESS entry in TCAM that has been specified as being able to accept messages from that program. This macro creates a logical output data set for the program.

Messages are moved into an application program work area by an application program GET or READ macro that specifies the *dcbname* of the logical input data set. (See Figure 2-1.) Messages being transmitted from an application program are transferred to TCAM by specifying the *dcbname* of the logical output data set in a PUT or WRITE macro in the application program.

## The Input Data Control Block (DCB) Macro

The DCB macro generates no executable code. When your program is assembled, each input DCB macro in the program is allocated space in main storage for a data control block. This data control block defines a single input data set for the program, which will consist of the messages or records being sent to an application program from a single TCAM destination queue. Each separate input data set to a particular program must be defined by its own input DCB macro.

When the application program executes, the values specified on the DCB macro, along with any values specified on the related MVS job control language data definition card, are integrated into the data control block.

The function of this control block is to provide parameters to TCAM and the operating system, defining the record format that the application program expects from this particular data set. The operating system then ensures that any data transferred to the program from this data set does indeed have the specified format.

The following operands may be omitted from the DCB macro in the application program and coded as parameters of the DCB operand of the job control language, data definition (DD) statement defining this logical data set. These operands, explained in Chapter 5 and in the *JCL Reference* manual, are BLKSIZE, LRECL, BUFL, OPTCD, and RECFM. If any of these operands are coded in the application-program DCB macro, the corresponding DD values specified will be ignored. Most DCB values can also be altered during program execution (see Chapter 5). For details on this capability, see the *Data Management Services* guide.

The input DCB macro allows the application programmer to specify:

- Whether data transfer from this data set is to occur using BSAM-compatible or QSAM-compatible request macros (MACRF)
- Whether the application program can handle entire messages or only message portions called segments (OPTCD)
- The format and characteristics of the input records (RECFM,OPTCD)
- The length in bytes of the message or record that this application program expects from TCAM for each GET or READ macro (LRECL)
- The length in bytes of the application-program work area into which input data is to be transferred (BLKSIZE)

- The length in bytes of the buffers to be used in TCAM to transfer messages from the TCAM message queue to the application program (BUFL)
- The label of an MVS data definition card that describes where this data set resides (DDNAME)
- The data set organization of this input data set (DSORG)
- The address of a list of user-written routines that perform such activities as checkpointing your program (EXLST)
- The address of a routine to be given control when the normal end of a series of data records is reached (EODAD)
- The address of a routine to be given control when a message overflow error on input occurs (SYNAD)
- Whether the EODAD exit should be taken when the MCP is halted (STOP).

For special considerations while coding an input DCB macro, see the section "Special Considerations for DCB Macro Operands" following in this chapter.

## The Output Data Control Block (DCB) Macro

If an application program generates any type of output, that output must be directed to a logical output data set. Therefore, an output data set will contain all the messages or records that are to be sent from an application program to the MCP. These messages will then be directed by TCAM to their destination. An output data set is defined by an output DCB macro. Messages are transferred from the application program to the MCP by a PUT or WRITE macro specifying the label of the appropriate output DCB macro.

The output DCB macro allows an application programmer to specify:

- Whether the application program is to transfer entire messages or only message segments to TCAM (OPTCD)
- The format and characteristics of records in the output data set (RECFM, OPTCD)
- The label of an MVS data definition card that describes where this data set resides (DDNAME)
- The data set organization of the output data set that this DCB refers to (DSORG)
- Whether data transfer to this data set is to occur in BSAM mode or QSAM mode (MACRF)
- The length in bytes of the message or record that TCAM expects from this program (LRECL)
- The length in bytes of the application-program work area from which data is to be transferred to TCAM (BLKSIZE)
- The length of the TCAM buffers used to receive messages from this application program (BUFL)
- The address of a routine to be given control if a logical output error occurs (SYNAD)
- The address of the problem-program exit list (EXLST).

## Special Considerations for DCB Macro Operands

There are several special considerations in coding a DCB macro for an application program in the TCAM environment. These considerations are discussed by operand keyword:

### BLKSIZE

In TCAM, the BLKSIZE operand is used to determine the size of your application-program work area. This work area must be large enough to hold a record of the length specified by the LRECL operand plus all the optional control fields specified by the OPTCD operand. This value (BLKSIZE) is used to calculate the length of all variable or undefined-length records that it will send to the application program. For undefined-length work units, this value (BLKSIZE) may be dynamically overridden on a work-unit by work-unit basis by the LENGTH operand of the READ macro.

### BUFL

Since message buffers for a TCAM system are acquired and allocated in the MCP, the application programmer does not need to be concerned about either buffer size or the number of available buffers. If, however, it becomes necessary for you to change the size of the buffers that the MCP uses to transfer messages to and from your application program, you may use this operand. Only buffer size may be altered with this operand; buffer availability is still handled automatically by the MCP.

You should code a value for the BUFL operand only if you wish to change the value specified in the MCP. Since a value coded here overrides a value coded in the MCP, this operand should be used only after careful consideration.

TCAM buffer concepts are discussed in Appendix A of this manual. You should review that material before coding a value here.

### DSORG

The only specification that can be made on this operand in the TCAM environment is DSORG=PS.

### LRECL

The LRECL operand must specify the exact length of the fixed-length or undefined-length record you expect to be transferred into or out of your work area. Contrast this value with the BLKSIZE operand, which must specify the size of the work area. This value must include any optional fields you have specified through the OPTCD operand.

**MACRF**

In the TCAM environment, the MACRF operand specifies not only the data movement macro instructions (GET, READ, PUT, or WRITE) that can be used in your program, but also the processing mode (MOVE or LOCATE) that is used to transfer your data to and from TCAM. In addition, you can specify on any input DCB macro whether you intend to use the TCAM POINT macro to retrieve specific messages from that input data set.

BSAM-compatible (READ, WRITE, CHECK) and QSAM-compatible (GET, PUT) request macros are supported in the TCAM environment. You should use whichever method suits your application requirements best. Refer to the data transfer discussion in Chapter 3 for details on the differences between BSAM-compatible and QSAM-compatible operation.

The differences between MOVE and LOCATE processing modes are also discussed in Chapter 3.

**OPTCD**

The OPTCD operand parameters have different meanings depending on the environment for which the DCB macro is coded. You will not be able to test any of the TCAM OPTCD operand parameters before implementing them because they have different meanings in the SAM environment for which they must be tested. Therefore, it is advisable to add these parameters to your program through the DCB operand on the appropriate JCL data definition card when you execute any program using OPTCD operand parameters.

**RECFM**

If RECFM=V or U, TCAM assumes the length of the program work area to be equal to the BLKSIZE operand value. If RECFM=F, TCAM assumes the length of the program work area to be equal to the LRECL operand value.

**STOP**

This input DCB macro operand must be coded in conjunction with the EODAD operand. If the EODAD operand is not specified, you must not code this operand. STOP allows you to extend the function of the EODAD operand beyond the normal end-of-data condition. It allows you to define the specific action you wish to be taken if an MCPCLOSE macro is issued or a TCAM system closedown is requested while your application program is executing. By specifying the STOP operand or either or both types of system closedown, you can provide for an orderly closedown under normal closedown conditions.

## SYNAD

The input and output DCB macros for all TCAM application programs permit you to specify an exit to be taken when certain errors occur during transfer of data between the MCP and the application program. This is called the SYNAD exit, and it is specified by the SYNAD operand of a DCB macro.

The SYNAD operand specifies the symbolic name of a subroutine that receives control when certain errors occur. This routine may be either open or closed and must be coded by the application programmer. This facility is offered so that you may continue processing instead of abnormally ending your job before processing has been completed.

The SYNAD routine is entered after a GET or CHECK macro is issued immediately following an error. In your SYNAD routine, you must close and reopen the input data set before issuing another GET or READ macro to it if one of these errors occurs, before TCAM will pass any more messages to your program. If one of these error conditions occurs and SYNAD is not specified, TCAM returns a completion code to the application program in register 15 following the next GET or CHECK macro. In this case, your main routine must close the data set.

The SYNAD routine specified by an input DCB macro is given control if: (1) message processing has been specified by coding OPTCD=U on this DCB, (2) the message to be transferred by the current GET or READ macro is larger than the work area available to it, and (3) no position field is specified for this work area (OPTCD does not also specify C).

The SYNAD routine specified by an output DCB macro is given control when: (1) the position field in the program work area (see next chapter) contains a value that is invalid, (2) the position field indicates that the current work unit is out of sequence (for example, the position field indicates that this is the first portion of the message, but the position field for the previous work unit did not indicate end-of-message for a previous message), (3) the name in the destination field in the program work area is not a valid entry in the MCP terminal-name table, (4) if the work-unit length exceeds the work-area length, or (5) if a TCAM quick closedown is in progress.

A TCAM SYNAD routine expects input identical to that provided by the QSAM and BSAM access methods for their SYNAD exits as explained in the *Data Management Services* guide. Therefore, both the SYNADAF and SYNADRLS macro instructions may be issued in your TCAM SYNAD routine. See the *Data Management Macro Instructions* manual for details on the function and use of these macros.

In designing a SYNAD routine, consideration must be given to the register and status indicator field contents on entry to the routine.

Input from TCAM to a SYNAD routine in an application program consists of several loaded registers. When control is transferred to the SYNAD routine, the general registers will be in the format described by Figure 2-2.

For BSAM (READ or WRITE) support the fifth word of the DECB associated with each read (DECB+16) contains a status-indicator address. Status indicators for the SYNAD routine are:

| **Address Offset** | | **Meaning** |
|---|---|---|
| Byte | Bit | |
| +2 | 0 | Command reject (work units out of sequence) |
| +13 | 1 | Incorrect length (work-area overflow) |

| **Register** | **Bits** | **Meaning** |
|---|---|---|
| 0 | 8-31 | Address of the data event control block (DECB) for BSAM-compatible programs; address of status indicators for QSAM-compatible programs. |
| 1 | 0 | Bit is on for error caused by GET or READ. |
| 1 | 1 | Bit is on for error caused by PUT or WRITE. |
| 1 | 4 | Bit is on if user specified an invalid destination (PUT or WRITE). |
| 1 | 8-31 | Address of associated data control block (DCB). |
| 2-13 | 8-31 | Contents before the macro instruction was issued. |
| 14 | 8-31 | Return address. |
| 15 | 8-31 | Address of the error analysis routine specified by the SYNAD operand of the input DCB. |

**Figure 2-2. Register Contents Upon Entry to the SYNAD Routine**

If the bits are on (that is, bit=1), then you have the error indicated. No other fields in the SAM-compatible status indicator field are used by the TCAM.

If you issue a SYNADAF macro in your error analysis routine, you will receive the following values in your SYNAD routine in the specified registers:

- Register 0 will contain a return code of X'00' right adjusted.
- Register 1 will contain the address of a buffer containing a message describing the TCAM/SAM error. The message consists of EBCDIC information and is in the form of a variable-length record such as that described in Figure 2-3.

| Bytes | Contents |
|---|---|
| 0-7 | SAM variable-(or variable-blocked) length prefix |
| 8-49 | (character blanks) |
| 50-57 | job name |
| 58 | ,(character comma) |
| 59-66 | stepname |
| 67 | ,(character comma) |
| 68-73 | (character blanks) |
| 74 | ,(character comma) |
| 75-82 | ddname (name of DD statement in which QNAME parameter is coded) |
| 83 | ,(character comma) |
| 84-89 | macro format (GET,PUT,READ, or WRITE) |
| 90 | ,(character comma) |
| 91-105 | error description (WORKAREA OFLOW, INVALID DEST, or SEQUENCE ERROR) |
| 106 | ,(character comma) |
| 107-120 | ************** |
| 121 | ,(character comma) |
| 122-125 | TCAM |

**Figure 2-3. Format of the SYNADAF Message Buffer**

## Special Considerations for Multiple TCAMs

An application program can open to only one TCAM.

If the application program is a subtask of the initiator, it can open only to the MCP attached as a subtask of that initiator.

If an application program is in a separate address space, the OPEN macro connects the application program to the TCAM with BASED=CVT specified or defaulted to on the INTRO macro. If there is no such TCAM active, the application program is abnormally ended.

Because there may be multiple TCAMs in a single host processor with OS/VS2 MVS SP-Version 1 Release 3.1 and Version 2 Release 1.1, you may select the TCAM that your application program will communicate with.

An application program can select a TCAM by qualifying the QNAME parameter on the DD statement in the JCL. The format is:

QNAME=procname.jobname

where *procname* is the TPROCESS entry in this TCAM, and *jobname* is the name of the TCAM job.

You do not need to specify a jobname in the following cases:

1. TCAM is not running in an OS/VS2 MVS SP-Version 1 Release 3.1 or Version 2 Release 1.1 system.
2. An application program is already opened to a TCAM.
3. There is only one TCAM in the host processor and it specifies or defaults to BASED=CVT on the INTRO macro.
4. The application program is a subtask of the initiator.

If you do specify a jobname when it is not required, the jobname will be verified. For the cases mentioned above, the OPEN macro will fail in:

- Case 1 if the TCAM is not active, of if the TCAM specified is different from the active TCAM.
- Case 2 if the TCAM specified is different from the TCAM that the application program is already opened to.
- Case 3 if the TCAM specified is different from the CVT-based TCAM.
- Case 4 if the application program is not a subtask of the TCAM specified.

## The Control Blocks in an MCP

The transfer of messages between the TCAM MCP and an application program is controlled by several parameters specified in a process control block (PCB). A PCB, then, is the MCP counterpart to an input or output DCB in an application program.

The process control block is created through expansion of a PCB macro. The PCB macro is coded by the TCAM system programmer since it must be included in the MCP. The PCB macro assigns a particular TCAM MH to a particular application program (see Figure 2-1). It also specifies the size of the buffers in the MH that will handle messages being sent to and received from the application program.

After a message whose destination is an application program is processed by the incoming group of MHs, it is sent to a message queue that has been defined by a TPROCESS macro. Expansion of the TPROCESS macro caused the name of the message queue for your application program to be included as an entry in the MCP terminal table. This allows TCAM to treat the queue for your program in relatively the same manner as it would treat a queue for an external LU; that is, TCAM can process and send messages to a queue for your program in the same manner it would send messages to a queue for an external LU.

Several operands on the TPROCESS and PCB macros are of interest to you. If the TCAM system programmer codes values for those operands, you should be aware of what those values are.

## PCB Macro

Keep in mind that even though you may not be responsible for coding these macros, you should be aware of what values are coded because they will affect how your application program functions.

The PCB macro operand parameters of interest to you are: SFLAG=YES, DATE=YES, BUFSIZE=nn, TIMEDLY=number, and PUTCNT=number.

***SFLAG=YES:*** A message segment on the destination queue for an application program is normally not marked serviced until the next GET or READ macro instruction has executed. Every time your application program terminates abnormally, the last message or message segment it received *might be lost* during termination. Therefore, you should allow for this possibility. Coding SFLAG=YES assures you that the last message obtained by a GET or READ macro will be marked as serviced by TCAM if your program terminates abnormally. If TCAM is terminating, the last message will not be marked serviced. See the discussion on message retrieval and queue reset facilities in Chapter 3.

***DATE=YES:*** If YES is specified here, it should also have been specified for DATE in the associated TPROCESS macro. This operand specifies that TCAM will record the date and time of receipt of every message addressed to this application program. When the application program issues a GET or READ macro, TCAM will place the time, date, and source of the message in a 16-byte work area specified by the DTSAREA operand of a TPDATE macro in your application program. This date and time will indicate the moment when the message was available for processing. In extended networking environments, this could be later than the time of entry into the network. If the earlier time is needed, that information is contained in the fixed header prefix (FHP).

***BUFSIZE=** nn:* This operand specifies the size of the TCAM buffers that will be assigned to handle messages for your application program. The value coded here in the PCB macro may be overridden by coding a different value in the BUFL operand of the associated DCB macro in your application program. If you do not code a value for the BUFL operand in your program, the buffers will be the size that is specified here.

***TIMEDLY=** number*, PUTCNT=*number:* These operands may be specified to create a time delay in the processing of PUT or WRITE macros to pace buffer utilization and to keep from flooding the queue. TCAM processes the number of PUT or WRITE macros specified in PUTCNT, then waits the number of seconds specified in TIMEDLY. This process is repeated until all PUT or WRITE macros have been processed. Both operands must be specified to activate the procedure.

## TPROCESS Macro

There are also several operand values that may be coded on the TPROCESS macro that should be of interest to you. The following operand parameters will have an effect on your application program: ALTDEST=entry, CKPTSYN=YES, RECDEL=delimiter, QBACK=YES, QUEUES=form, SECTERM=YES, SECURE=YES, and DATE=YES. In addition, the PCB operand must specify the name field of the PCB macro related to this message queue and your application program. See Figure 2-1.

*ALTDEST=entry*: If this TPROCESS macro relates to an input message queue in your program, this operand must specify an alternate queue to which messages can be sent while this queue is being reorganized. If this operand is omitted and reusable queues are specified in the MCP, messages may be overlaid and lost during queue reorganization.

If this TPROCESS macro relates to an output message queue in your program, this operand must specify the destination to which replies to operator commands issued from this application program are to be sent. The destination may be this program, an external LU named by a TERMINAL macro, or an application program represented by another TPROCESS macro.

Response messages can also be sent to the destination specified by the RSPDEST operand of the CODE or the IEDOPCTL macro. If neither ALTDEST nor RSPDEST is specified, response messages go to the dead-letter queue (DLQ).

***CKPTSYN=YES:*** YES must be coded if you wish checkpointing of the application program to be synchronized with checkpointing of the MCP. For more detail see checkpointing in Chapter 4.

***RECDEL=*** *delimiter:* This operand defines a one-byte, nonzero hexadecimal value that TCAM will use to delimit every *record* (not message) that it sends to your application program in response to the execution of a single GET or READ macro. You may want to set up the application program to test for this particular delimiter.

***QBACK=YES:*** If YES is coded, your application program may issue the QRESET macro. The QRESET macro cannot be issued if this value has not been coded. For details, on the QRESET macro and the queue reset facility see the discussion in Chapter 3. QBACK=YES causes a module to be loaded into the MCP region and a work area to be allocated to support retransmission of messages from a specific message queue at a particular output message sequence number.

***QUEUES=DR, DN, MO, MR, or MN:*** This operand specifies where the message queues containing messages for this application program are to be maintained (for GET or READ operating only). If the type of the data set specified by this operand does not correspond to a related message queues data set defined by a DCB macro in the MCP, the TCAM MCP will abnormally terminate. By omitting the QUEUES operand, the user specifies that this process entry is for PUT or WRITE operations only. Therefore, this operand must not be coded for PUT or WRITE operations.

See the TPROCESS macro section in *TCAM Installation Reference* for details.

***SECTERM=YES:*** Unless YES is coded here, your application program may not issue basic operator control commands. See the operator control command section in Chapter 4.

***SECURE=YES:*** If YES is coded, your application program cannot open the input DCB until the system operator replies to a console message. If the operator does not allow the input DCB to be opened, your program will ABEND with a system completion code of hex 043 and a hex 06 in register 15.

***DATE=YES:*** If YES is specified here, it must also have been specified in the associated PCB macro. This operand specifies that the MCP record the time and date of receipt of every message addressed to this application program. When the program issues a GET or READ macro, the MCP will place the time and date, and the source of the message in a 16-byte work area specified by the DTSAREA operand of a TPDATE macro in your application program.

## Coding TCAM Application Programs

Before you code an application program, you should be familiar with the first four chapters of this manual and with the instructions in the *TCAM Installation Guide* for coding DCBs, PCBs, and the LOCK macro because severe errors can result from misuse or omission. When you start writing the application program, work with a system programmer to code the MHs for the MCP. At first, use only the code necessary to establish the TCAM interface and to test the transfer of messages or data between the application program and the MCP. After successful testing of the interface, you can then add more code and function.

### Steps for Message Flow

The following coding steps are necessary to implement message flow between TCAM and an application program:

- Specify both input and output DCB macro parameters in the application program.
- Code an OPEN macro in the application program for each DCB you specify.
- Check that the appropriate QNAME operands and jobname qualifiers are in the JCL that specifies execution of the application program.
- Check that the appropriate PCB-macro and TPROCESS-macro parameters are coded in the MCP.
- Request the actual transfer of message data between the application program and TCAM by issuing a GET, READ, PUT, or WRITE macro in the application program.

## Use of Data Management Macros

Every TCAM application program must include the data management macros, such as DCB, OPEN, GET, PUT, or CLOSE, that are necessary to provide interaction with TCAM. Although not required for implementing message flow, every application program should include a CLOSE macro to deactivate each of the program's data sets upon completion of processing. Failure to include a CLOSE macro causes an error if you try to end execution of an application program while a data set is open.

An application program may be coded in Basic Assembler Language, COBOL, or PL/I. The language used must have the ability to generate QSAM or BSAM input/output requests.

All of the TCAM-compatible data management macros that may be used in an application program are listed at the beginning of Chapter 5. Any macro which may be issued in an MVS problem program may be issued in a TCAM application program. When a TCAM macro is encountered during execution in a system without TCAM, a return code, indicating the condition, is set and control immediately passes to the next sequential instruction. Otherwise execution of the program is not changed.

With one exception, TCAM macro instructions issued in an application program execute as specified *only if* the task in which they are issued is initiated with JCL that contains at least one DD card specifying a QNAME operand.

The exception is that you may issue a GET, READ, PUT, or WRITE macro from an application program that is an attached task provided the attached task is also a TCAM application program that was initiated with proper JCL coding. For example, the attaching program or task can open the necessary data control blocks required to establish the MCP-application program interface and issue a GET, READ, PUT, or WRITE macro to a TCAM destination queue. If you code your program to take advantage of this exception, an attached task or application program can communicate with the MCP without a special interface.

To TCAM, an application program is similar to an external LU. As a valid destination for messages, it must have a destination queue to which the GET macro is issued. The location of the queue is specified by the QUEUES operand of the TPROCESS macro. The QNAME parameter on the DD card specifies the name of the process entry with which the destination queue is related. Further information about the QNAME parameter can be found in "Data Definition (DD) Statement Parameters" on page 2-18.

One TPROCESS macro in the MCP should be defined for each queue used by an application program: one TPROCESS macro, for GET or READ, and one TPROCESS macro, for PUT or WRITE. More than one TPROCESS macro can name the same PCB. The GET or READ TPROCESS macro must specify the QUEUES operand, and the PUT or WRITE TPROCESS macro must not specify the QUEUES operand.

## Secured Queues

In addition, the TPROCESS macro allows a queue to be specified as *secured*. A secured queue is one that can be opened only after authorization from the system operator. If you want a queue to be secured, code SECURE=YES on the GET or READ TPROCESS macro. When a program attempts to open a secured queue, a write-to-operator-with reply (WTOR) is issued. The WTOR gives the operator the name of the job attempting to open the queue and the name of the queue. The operator can then either release the queue or abort the job.

## Activating an Application Program

An application program can run as a separate job, a subtask of the MCP or as a subtask of the TCAM initiator, but in each case, it must have a priority lower than that of the MCP. If the application program runs as a subtask of the MCP, the priority can be lowered by the CHAP macro. Whether the application program runs as a separate job or as a subtask, all application programs must follow standard linkage conventions in saving and restoring the registers of the calling program.

## Deactivating an Application Program

An application program running as an attached subtask of the initiator can be closed using initiator commands. If the application program runs as a separate job, the system operator can close it with a Cancel command. However, if it runs as a subtask of the MCP, you must close it some other way because the Cancel command cannot locate the application program. One way to close an attached subtask is to have the application program test for a special closedown message sent to it by an external LU and to branch to a closedown routine when it receives this message. The STOP operand, discussed earlier in "The Input Data Control Block (DCB) Macro" on page 2-3 specifies the type of closedown being performed. For debugging purposes, it is better to run the application program separately to avoid affecting the MCP.

Your system will work more efficiently if you assure that the work-unit size in the application program is compatible with the buffer size in the MCP.

If you code any non-TCAM macros or use the SYNAD operand of a DCB macro or any other exit, read the appropriate publications. This manual covers only coding considerations for TCAM application programs.

## Installation Procedures, Assembly, Linkedit and Execution Considerations

The installation procedure for TCAM V3 is different from that of previous releases. TCAM V3 requires the system programmer to establish two new release unique libraries that isolate the product from the normal system libraries. The two new libraries contain the macros and modules constituting the product. JCL modifications for all jobs relating to TCAM V3 is also required.

Assume these data set names for the release unique libraries.

|  |  |
|---|---|
| Macros - | SYS1.TCAMMAC |
| Modules - | SYS1.TCAMLIB |

The following examples demonstrate JCL requirements for using TCAM V3 libraries and does not constitute the complete JCL stream.

### Assembling a TCAM Application Program:

```
//TYPASM     JOB     Job card parameters
//STEP1      EXEC    ASMFC
   .
   .

//ASM.SYSLIB  DD      DSN=SYS1.TCAMMAC,... TCAM V3 macro library
//            DD      DSN=SYS1.MACLIB,...  Macro library
```

*Note:* The concatenation of the new macro library in front of the normal system macro library. This construction of the JCL stream allows the TCAM V3 macro expansion to be retrieved from the TCAM V3 library.

### Link-Editing a TCAM Application Program:

```
//TYPLKED    JOB     Job card parameters
//STEP1      EXEC    PGM=IEWL
   .
   .

//SYSLIB     DD  DSN=SYS1.TCAMLIB,...  TCAM V3 module library
//SYSLMOD      DD  DSN=SYS1.TCAMLIB(member),... new load module
                                  (optional)
```

*Note:* The library assignments for SYSLIB and SYSLMOD. This JCL stream allows the link edit process to retrieve the TCAM V3 modules for inclusion into the final load module. The SYSLMOD library replacement is optional. However, this mechanism keeps TCAM V3 modules on one library rather than combine them into the LINKLIB which may also contain TCAM V2 modules that have been duplicated for the TCAM V3 environment.

### Executing a TCAM Application Program:

```
//TYPEXEC    JOB     Job card parameters
//STEP1      EXEC    PGM=name,...      name could be IEDQTCAM
//STEPLIB    DD   DSN=SYS1.TCAMLIB,...  TCAM V3 module library
   .
   .
   .
```

*Note:* In the above example a STEPLIB card was added which concatenates this library in front of the normal libraries search at module location time.

## Testing TCAM Application Programs

For development and debugging purposes, TCAM allows you to test a new application program in a BSAM/QSAM environment until it executes without error before running it in conjunction with a TCAM MCP. For example, you can test the logic of a TCAM application program by using input from a card reader with output going to a printer. In many cases, you can convert from a test environment to a TCAM environment merely by changing the DD statements for the application program data sets.

If you wish to test a TCAM application program in a BSAM or QSAM environment, you must remember the following points:

- Depending on the environment, the OPTCD operand of the DCB macro has incompatible meanings. See the appropriate JCL Reference manual for the differences between TCAM and non-TCAM DCB parameters. This operand should be omitted from the DCB macro when testing and specified if needed at execution through the DCB parameter of the appropriate DD statement. (Refer to "Special Considerations for DCB Macro Operands" on page 2-5 for details on DCB operands).

- Test data for the test environment should contain any optional fields that would be present in the work area of the program if it were run under control of TCAM. (See the work area section in Chapter 3.)

- The POINT macro may be issued but it has a different meaning in the BSAM/QSAM environment than in a TCAM environment. (See "Retrieving Messages from TCAM Disk Data Sets (the Point Macro)" on page 3-30 Chapter 3 for details).

- When issued in a BSAM/QSAM environment, the TCOPY, QCOPY, TCHNG, and MCPCLOSE macros place a return code in register 15 indicating that TCAM is not in the system, and immediately pass control to the next instruction. (Refer to Chapter 4, "Optional TCAM Facilities for the Application Programmer" on page 4-1 Chapter 5, "TCAM Application Programmer's Macro Reference Guide" on page 5-1 for details.

- The DCB checkpoint exit specified by the EXLST operand on the DCB macro is ignored in a BSAM/QSAM environment. (See the DCB and checkpoint discussions in Chapters 2 and 4).

- TCAM and BSAM/QSAM use different return codes. Therefore, register 15 cannot be tested for the same codes after a GET or PUT macro for an application program when running in a BSAM/QSAM environment. (See the GET/PUT macro discussions in Chapters 2 and 5 and in the *Data Management Macro Instructions*.)

By allowing for each of these points, you can plan, code, and test all of your application programs in a controlled environment until they execute without error, and then integrate them into the TCAM environment. You should be able to implement most programs without even reassembling them.

## Data Definition (DD) Statement Parameters

Every input and output DCB has a DDNAME operand that must be given a parameter. The parameter specified for this operand must also be specified as the symbolic label on a job control language data definition (DD) card that identifies the respective logical data set to the operating system. This DD statement must also include a QNAME operand that specifies the symbolic name of a TPROCESS macro in a TCAM MCP and may also specify the name of the TCAM job.

The TPROCESS macro defines a particular message queue (a TPROCESS queue) in TCAM that will either provide input to or accept output from your program. The TPROCESS macro also identifies a specific MH through its PCB operand. Thus, the data definition (DD) card helps connect an application program DCB to a specific TPROCESS queue. At application program execution, your job control language must specify one data definition statement for each DCB in the program.

See the appropriate *JCL Reference* manual for the details on coding DD statements. The format for the DD statement is:

//ddname DD QNAME=procname.jobname

*ddname* is the symbolic name of this data definition statement, and must be identical to the name specified in the DDNAME operand or a related input or output DCB macro in the application program.

If the *DDNAME* of the DD statement in the TCAM job step containing the QNAME=operand matches a host LU name, the OPEN for the host LU's ACB will fail. In a TCAM V2 environment, such a situation indicated that TCAM should provide subsystem support; TCAM V3 does not support subsystems. This restriction is most likely to be encountered in an environment where TCAM applications are attached by the TCAM initiator.

*procname* is the symbolic label of a TPROCESS macro in a TCAM message control program that specifies a process control block for an application-program input or output queue in TCAM. This value may be easily changed at execution by changing DD cards, thus allowing the user to specify different queues at different times.

*jobname* is the name assigned to the particular TCAM that is to contain the specified *procname*. For more information on *jobname* see the section titled, “Special Considerations for Multiple TCAMs” in this chapter.

Several DCB macro operands may be omitted from the DCB coded in the application program and may be coded as parameters of the DCB=operand of the DD statement. These operands are described in Chapter 5 and in the appropriate JCL Reference manual. The operands are BLKSIZE, LRECL, BUFL, OPTCD, and RECFM. If any of these operands are already coded in the application program, however, these values on the DD statement will be ignored.

## Starting and Stopping the Application Program Interface

Communication between the TCAM MCP and an application program cannot begin until both the MCP and the application program are running. Activation of both the MCP and the application program may be handled by any one of several methods.

### Starting TCAM Application Programs

Both the TCAM initiator and TCAM application programs may be started as normal jobs through the system card reader. They may also be started by issuing a START command through the system console naming a cataloged procedure. The cataloged procedure must specify the TCAM initiator or the application program that is to be executed. Once the TCAM initiator has been started, it attaches the TCAM MCP as a subtask.

Both of the above procedures require manual intervention with the operating system; thus, some user-created operating procedure must be implemented in order to notify the operator when (or how often) he must execute the procedure. These two methods also require that the application program begin execution in a different address space than the TCAM MCP.

A third method applies only to application programs and does not require operator intervention. The application program may be activated by the TCAM initiator, just as a system service program is activated. The application program will then begin executing as a subtask in the same address space as the MCP. Both the MCP and the application program will be executing as subtasks of the TCAM initiator. For more information on how to treat an application program as a user-supplied system service program, see titled “Initiating and Terminating TCAM” in *TCAM Installation Guide.*

Application programs that are treated like user-supplied system service programs may be automatically activated and deactivated with the MCP.

Remember that an application program runs asynchronously with the MCP. The accepted procedure for activating an application program is to start the TCAM MCP, and then start your application program.

## Opening and Closing TCAM-Related Data Control Blocks (The OPEN and CLOSE macro)

Opening of data control blocks in your application program is implemented by the OPEN macro. Closing of data control blocks in your application program is implemented by the CLOSE macro. See *Data Management Services* for a discussion on modifying data control blocks.

The actual activation and deactivation of the interface between an application program and the TCAM MCP is handled by OPEN, CLOSE and MCPCLOSE macro instructions in the application program. The OPEN and CLOSE macros for TCAM application programs are coded and used in the same way as OPEN and CLOSE macros coded for BSAM and QSAM data sets in a local (BSAM/QSAM) environment, as described in *Data Management Macro Instructions*. No other TCAM-related macro instruction in an application program may be successfully executed before at least one of the input and output data set data control blocks in the application program is opened.

Both the list and execute forms of the OPEN and CLOSE macros may be coded. You may also code any option for the OPEN and CLOSE macros shown in *Data Management Macro Instructions* to run an application program in a BSAM/QSAM environment and for debugging purposes; but when the program is executed in a TCAM environment, the operands not applicable to TCAM are ignored. As stated previously, this facility allows you to develop and test TCAM application programs in a BSAM/QSAM environment, and then integrate the programs into the TCAM environment without recompiling.

More than one data set may be opened or closed with the same OPEN or CLOSE macro. The details for coding OPEN and CLOSE macros in a TCAM application program are discussed in Chapter 5.

## Stopping the TCAM Message Control Program (the MCPCLOSE Macro)

Stopping of the TCAM MCP may occur either in conjunction with an MCPCLOSE macro or through the use of the Closedown of MCP (HALT) operator command.

Closedown may be effected by an MCPCLOSE macro issued as part of a termination routine in an application program. One procedure is to send a special closedown message to the MCP from an external LU or application program. This message would be directed to each active application program in the system by specifying the names of the appropriate terminal-table process entries as destinations. Each application program would contain a user-written termination routine that would be activated when the message was received. The termination routine might perform the following steps:

1. Issue an MCPCLOSE macro
2. Close any open application program data sets
3. Issue an MVS RETURN macro in order to end the application program job.

If the extended operator control system service program is running in conjunction with the MCP, the Stop Application Program extended operator command may be used to trigger closedown of one or more application programs.

In order to stop a single application program, the operator entering this command specifies the name of a GET/READ TPROCESS macro for the application program that is to be stopped. More than one application program may be stopped by specifying the name of a TLIST macro instruction defining a distribution list. The list contains the name of a GET/READ TPROCESS macro for each application program to be stopped.

Upon receiving the CLOSE command, the system generates a close message and queues it to the specified application program or programs. To use this facility, each application program must be able to recognize the message based on the message text, which is CLOSE. The message text may or may not be preceded by a fixed header prefix (FHP). If it is, the beginning of the message text may be found by adding the offset from the beginning of the FHP, contained in the field FHPHEADP, to the address of the application program's input work area. When the application program receives a CLOSE message, it must close its data sets and return control.

At least two levels of priority should be coded for the TPROCESS queues to ensure that the CLOSE message gets to the application program without delay.

The system programmer may force a closedown by coding the SETEOF macro in his AMH coded to execute when the closedown message is processed. See the *TCAM Installation Reference* for a description of the SETEOF macro. When the application program receives the message on which SETEOF has executed, access method code associated with the application program branches to the address specified by the EODAD operand of the input DCB macro when the next GET or CHECK macro is issued.

An operand of the MCPCLOSE macro allows you to specify either a quick or a flush closedown. The difference between a quick and flush closedown is discussed in the MCPCLOSE macro discussion in Chapter 5. After all message traffic has ceased, TCAM checks for open application-program data sets; when all such data sets are closed, control passes back to the MCP. TCAM then starts a user-coded routine in the MCP that issues a CLOSE macro for each open data set in the MCP and ends with an MVS RETURN macro. This causes control to be returned to the TCAM initiator, which sends a WTOR to the system console, allowing the operator to either start another MCP or free the address space by terminating the initiator task.

If closedown is done via a Closedown of MCP operator command, any application program data sets that are open at the time message traffic ceases will cause an error message to be directed to the system console; the error message lists the open data sets for that application program. If more than one application program has open data sets, the message for the second application program will not be sent to the console until all data sets for the first program are closed. When all data sets are closed, the MCP is deactivated.

The Closedown of MCP operator command or the MCPCLOSE macro will also allow you to close all open application program DCBs by forcing the GET/READ routine to take the EODAD exit for each open input DCB. This will be done only if you code the STOP operand and the EODAD operand on each input DCB macro. See the section on data control blocks earlier in this chapter for a discussion of the STOP operand. An error message may still be issued if your EODAD exit routine is not a closed routine.

*Note:* When issuing the READ macro without issuing the CHECK macro, at least one complete message must be read before the EODAD exit may be taken.

## Summary

In review, each GET/PUT or READ/WRITE macro issued in an application program must specify the name of a data control block created by a DCB macro issued in that program. The DCB macro specifies (by its DDNAME operand) a job control language DD card label. The QNAME operand of the DD card defining each DCB to the operating system names a TPROCESS macro in the MCP. The PCB operand of the TPROCESS macro creating this entry specifies a TCAM process control block. The MH operand of the PCB macro creating the process control block specifies the MH that handles messages directed to or received from the application program.

After all the control block relationships are established, the control blocks must be opened before they can control the transfer of data between the TCAM MCP and your application program. This is accomplished through the use of the OPEN macro.

At this point, both TCAM and the application program are prepared to accept and process messages. Chapter 3 discusses the format and transmission of data between TCAM and an application program.

# Chapter 3. Transferring Data Between TCAM and an Application Program

This chapter discusses:

- The details of the procedure that TCAM uses to pass data to and receive data from application programs
- Messages, message segments, and records
- The TCAM work unit
- Application-program work areas and the data transfer macros
- Message retrieval and retransmission.

This chapter contains programming details that may not be understood by someone without a programming background.

## An Example of Message Flow Through TCAM

The following example is given to help you understand how a message flows through the TCAM system. The example is followed by a more detailed discussion concerning the specification of work units and work areas and the transfer of data.

A TCAM MCP is executing in one address space of your host processor. At 3 p.m., an operator at an external LU in your TCAM system initiates a transmission to the host. TCAM facilitates this transmission by helping the external LU establish a communication path through the VTAM (access method) residing in your host processor. The external LU transmits a five-thousand-byte message. The message contains five hundred accounts payable records and is addressed to the accounts-payable application program at the host. The accounts-payable program is not currently in main storage; it is not scheduled to begin until five p.m. The first network component to receive this message is a communication controller. The communication controller passes the message to VTAM in the host in several parts or segments. VTAM then passes these parts or segments to TCAM in the host. The TCAM MH examines the message header, which is contained in the first segment of the message. As a result of this examination, TCAM stores the entire message on a queue that can be addressed by the accounts-payable application program. At 5 p.m., an operator starts the accounts-payable application program.

Between the time that the first external LU sent a message at 3 p.m. and 5 p.m., several other external LUs have also sent in messages to the accounts-payable program, and the MCP has added these messages to the same destination queue as the first message. As the program completes processing each message, it generates a credit reply to the originating external LU before going on to process the next message. This credit message is accepted by the MCP, processed, and put on a destination queue for the respective external LU. If the external LU is available to receive messages immediately, the message is sent. If the external LU is not ready, the message waits on the queue until the external LU can accept it.

Now let's look a little closer at the procedure that occurs within the TCAM MCP at the time the message is received. If you don't understand some of the terms in the following description, refer to the glossary in the back of this manual. The primary intent of this discussion is to give you a general idea of what TCAM does to a message before it is passed on to an application program.

When a message is received from an external LU a TCAM MH examines the message header information in that message to determine the message destination. The message is processed by TCAM and placed on a destination queue for the destination.

Subsequently, the message is read from the destination queue and handled by the outgoing group of a second MH. In our example, this is the MH for the accounts-payable application program.

At this point, the message segment is put on the read-ahead queue. The destination queue may be on disk storage or it may be in main storage. The application program obtains messages or message segments from the read-ahead queue by issuing GET or READ macro instructions. These macros obtain the message in sections of data called work units. A work unit can be a message, a message segment, or a record that is the same size or smaller than the input area of the application program called the work area. The message or work unit is placed in the work area for processing by the program.

After processing the message, and assuming that a reply is necessary, the application program issues a PUT or WRITE macro to return the reply to TCAM. The reply is processed by the incoming group of the MH assigned to this application program (in our case the accounts payable application program). The message is then placed on a destination queue for the external LU. It is then processed by the outgoing group of the MHs assigned to the external LU and transmitted to the destination terminal via VTAM. This example shows message flow between an external LU and an application program; however, an application program can also send messages to another application program in the same way.

This is one example of how TCAM would function to process a particular application. How you decide to solve your application problem depends on the application. Before you begin coding, however, you will need to know a little more about messages, message segments, records, work units, and the work area. A few pointers on how to code the data transfer macros will also be helpful.

## The TCAM Inquiry/Reply Facility

An *inquiry* is assumed to be a message originating from an external LU, that is transmitted to an application program in the host. A *reply* is an immediate response from the program back to the external LU and should be logically related to the *inquiry* TCAM provides a means of achieving *inquiry/reply* interactions between an external LU and an application program, called *lock mode*.

*Message lock* involves establishing a logical connection between an external LU and an application program so that the external LU receives a reply to its inquiries. *Extended lock* maintains the lock capability over interaction of multiple inquires.

## The TCAM Work Unit

A *work unit* is the amount of data transferred between TCAM and an application program by a single READ, WRITE, GET, or PUT macro. A work unit must fit within an application program work area.

The way in which TCAM determines how much data to send to an application program via a single READ or GET macro depends upon the type of work unit *(message* or *record*) and the format of the work unit (fixed, variable, or undefined length) that you have specified on the associated DCB.

### Work Unit Types (Messages and Records)

An application program can only handle one work unit at a time. A work unit may be a record, a message, or a portion of a message called a message segment; a work unit that is a message or message segment may be, but need not be, a record.

In the TCAM environment, a *record* is a logically related set of data, the length of which is defined in the application program DCB macro LRECL operand. A record is the minimum amount of logically related data that can be sent or received by a single READ, WRITE, GET, or PUT macro. A record can, but is not required to, be delimited by a record delimiter. This delimiter can be added to a record at several points. It can be added at the message source, it can be added by a MH in the TCAM MCP before the record is transferred to your program, or it can be added by your program before transferring the record to TCAM. The primary purpose of a record delimiter is to clearly define a work unit as a record. The size of a record need not coincide with the size of a message or message segment; one message may contain many records. The delimiting character is ignored for fixed-length records.

A message may be transmitted as a single physical block or unit, or in several physical blocks called message segments. A message segment is that portion of a message that will fit in a single TCAM message buffer and be transmitted by a single GET, PUT, READ, or WRITE macro.

The advantages of specifying one type of work unit over the other depends on how you intend to process your data and on what other data management options you choose. If you want to process data in the distinct units that are the same size or smaller than your program work area, you should specify *record* processing. TCAM will pass on a clearly identifiable unit of data to your program for each GET or READ macro.

If your program can handle data in a more informal manner, you may want to specify *message* processing. In this case, TCAM will completely fill your program work area each time you issue a GET or READ macro.

Some of the differences between message and record processing on input or output from your program are:

| | **Message Processing** | **Record Processing** |
|---|---|---|
| On input to an application program | ● When a GET or READ macro is issued, ACF/TCAM brings in data until either the end of the message is encountered or the work area is filled. | ● When a GET or READ macro is issued, TCAM brings in data until (1) the delimiting character specified by the RECDEL operand and of the TPROCESS macro that defines the input data set for this program is encountered, (2) the end of the message is encountered, or (3) the work area is filled. |
| | ● If the work area has been filled and the end of the message was not reached, TCAM either brings in the next segment of the message with the next GET or READ macro (if OPTCD=C is specified), or goes to the error-handling routine specified by the SYNAD operand of the input DCB macro. | ● If the work area is filled, TCAM assumes that a complete record has been received. |

| | Message Processing | Record Processing |
|---|---|---|
| On output from an application program. | ● Whenever a PUT or WRITE macro is executed, TCAM transfers one work unit of data from the application program work area to the MCP.<br>● Same as for message-one work unit (record) is transferred per PUT or WRITE macro. | |
| | ● The RECDEL operand of the TPROCESS macro is ignored. | ● The delimiting character specified by the RECDEL operand of the TPROCESS macro is placed by ACF/TCAM at the end of each outgoing undefined-length record and each outgoing variable-length record. |
| | ● If a position field is present in the prefix of the work area and it indicates an initial or intermediate segment, TCAM transfers the next segment of the message to the MCP when the next PUT or WRITE macro macro is executed for this output data set. If no position field is present, TCAM assumes that the end of the current message coincides with the end of the work area. | ● If a position field is present TCAM considers all records to be part of the same message until the position field indicates that the current record is the last record in the message. If no position field is present, execution of the CLOSE macro in the application program for the output data set signals an end of the message. |

## Specifying Record or Message Processing

The OPTCD operand of the DCB macro associated with the GET, READ, PUT, or WRITE macro in your program allows you to specify whether you prefer message processing or record processing. If OPTCD=U is specified, either message or message segment processing is assumed. If U is omitted when specifying the OPTCD operand parameters, record processing is assumed. (See Figure 3-3).

### Processing a Message

When U is coded in the OPTCD operand of the input DCB macro, TCAM attempts to fill the work area with an entire message when a GET or a READ macro is executed. If the work area is not large enough for the entire message, TCAM transfers as much of the message as will fit in the work area.

If the entire message does not fit into the work area, the next portion of the message will be moved to the work area the next time a GET or READ macro is issued. This continues until the entire message has been processed. This is message-segment processing. Portions of messages processed in this way are not considered to be records since message processing rather than record processing was specified in the DCB macro.

To determine whether an incoming message will fit into the work area, TCAM must first know the length of the work area. For fixed-length messages, TCAM assumes the length of the work area to be equal to the number of bytes specified in the LRECL operand of the input DCB macro. For variable- and undefined-length messages, TCAM assumes that the work-area length is equal to the number of bytes specified in the BLKSIZE operand of the input DCB macro.

If a completed message does not fit into the work area with one GET or READ macro, TCAM takes one of three actions depending on what operands of the input DCB macro are coded. If the first or intermediate position field option is specified in the position field in the work area (see the discussion on "Defining Optional Fields in the Work Area" following), the portion of the message that did not fit into the work area on the first GET or READ will be obtained when the next GET or READ macro is executed. A PUT or WRITE macro may be issued before the second GET or READ macro. (See the OPTCD discussion for the input DCB macro in Chapter 5.)

If message processing had been specified, but no position field is specified on the DCB macro, TCAM gives control to the routine specified by SYNAD.

If neither a position field nor a SYNAD exit is specified, TCAM places a return code in register 15 and returns control to your program. This return code indicates that an error condition exists, and that you should probably terminate message processing and correct the error.

Work-area overflow may occur if the LRECL operand does not correspond to the work-area size. When a work-area overflow error occurs, TCAM discards the message that caused the overflow. If the input data set is closed and then reopened as a result of work-area overflow, the first message received in the work area following reopening of the data set is the message after the one that caused the overflow.

If U is specified in the OPTCD operand of an output DCB macro, TCAM performs message processing on any output associated with this DCB. If a position field is specified in the work area (by coding OPTCD=C in the output DCB macro), TCAM uses this field to determine whether the work area contains an entire message or only a segment of a message. If the work area does not contain an entire message, TCAM treats each piece of data moved from the work area by a PUT or WRITE macro as part of the same message until the position-field byte indicates that the work unit currently being processed is the last unit in the message. If no position field is specified, TCAM assumes that the entire message is currently in the work area. For message output, the application program is responsible for assuring that the position field contains the correct code.

Depending upon the format of the work unit (whether it is fixed, variable, or undefined), TCAM looks in the SAM-prefix (another optional field preceding the work unit) or in an output DCB field for the length of the outgoing work unit, and sends out the quantity of data specified in the appropriate field, allowing for optional fields in the work area. See Figure 3-1 for an explanation of how TCAM determines work-unit size on input and output.

## Processing a Record

If U is not coded in the OPTCD operand of the input DCB macro, TCAM treats each incoming work unit as a record, rather than as a message or a message segment. After an incoming message is placed on a TCAM queue for a specific application program, the application program should obtain the records in that message one at a time, with one record being passed for each GET or READ macro issued. The decision concerning the type of processing you wish to use depends on the size of the input message, the size of the records in that message, the size of the TCAM buffers, and the type of data in the message.

If you specify that the input to your application program is to be fixed-length records (by coding RECFM = F and omitting OPTDC = U on the input DCB macro), TCAM assumes that each incoming work unit is to be a record equal in length to the number of bytes specified in the LRECL operand of the input DCB macros (minus the length of any optional fields in the work area), and moves this number of bytes into your work area each time a GET or READ macro is executed against the data set defined by this DCB. The last record in a message may be shorter than the number of bytes specified by the LRECL field, in which case TCAM brings in the remaining number of bytes in this message.

If fixed-length records are designated as the output from an application program (by coding RECFM = F in the output DCB macro), each time a PUT or WRITE macro is executed TCAM transfers to the MCP a record equal in length to the number of bytes specified in the LRECL operand of the DCB defining the output data set, after making allowance for the length of optional fields in the program work area.

| Work-Unit Format: | | | Record | | | Message | | |
|---|---|---|---|---|---|---|---|---|
| | Work-Unit Type: | | Fixed | Variable | Undefined | Fixed | Variable | Undefined |
| Input side (GET/READ) | Work-unit size determined by: | LRECL field of input DCB | ● | | | ● | | |
| | | BLKSIZE field of input DCB | | ● | ● | | ● | ● |
| | | Length field of READ macro | | | ● | | | ● |
| | | Delimiter as specified in TPROCESS macro | | ● | ● | | | |
| | | End-of-message | ● | ● | ● | ● | ● | ● |
| | Work-unit size stored in: | Field in SAM prefix | | ● | | | ● | |
| | | LRECL field of input DCB | | | ● | | | ● |
| Output side (PUT/WRITE) | Work-unit size determined by: | LRECL field of output DCB | ● | | ● | ● | | ● |
| | | Length field of WRITE macro | | | ● | | | ● |
| | | Field in SAM prefix | | ● | | | ● | |
| | | Delimiter as specified in TPROCESS macro inserted after each record | | ● | ● | | | |

**Figure 3-1. Work-Unit Size Determination Chart**

If you specify that the input to your application program is to be variable- or undefined-length records (by coding V or U, respectively, on the RECFM operand of the input DCB macro), TCAM determines the length of incoming records according to the following requirements:

- If the delimiting character (specified by the RECDEL operand of the TPROCESS macro creating the destination queue addressed by the GET or READ macro) is encountered while the work area is being filled, TCAM assumes that the current record ends with this character. You may request that delimiting characters be removed from the data by specifying DELETE=YES on a TPDATE macro in your program.
- If the end of a message is reached before the work area is filled, TCAM assumes that the last character in the message is also the last character in the current record.
- If neither a delimiter nor the end of the message is reached by the time the work area is filled, TCAM assumes that the length of the record is equal to the size of the work area, minus the size of any optional fields in the work area. TCAM determines the size of the work area by looking in the BLKSIZE operand field of the input DCB.

When record processing is specified in the DCB macro defining an output data set, TCAM expects a single record for each PUT or WRITE macro issued to that data set. For variable-length records, the size of the record is indicated in the SAM prefix of the work area. For undefined-length records transferred by a PUT or WRITE macro, the size of the record is indicated by the LRECL field of the output DCB or an operand of the WRITE macro. The application program is responsible for filling the appropriate field with the appropriate value or control character so that TCAM will know how to process each work unit.

The differences between message and record processing are summarized in the section title “Work Unit Types” in this chapter.

## Work-Unit Formats

In addition to specifying work unit type, you must also specify the format of the work unit that your program is designed to process. You may specify a fixed-length record format, a variable-length record format, or an undefined-length record format. Your choice of format must be specified through the RECFM operand on the input and output DCB macros for the application program. These operands indicate whether the work unit will always be the same length (fixed), or whether they may vary in length from message to message or record to record (variable and undefined).

If messages or records sent to an application program may vary in length, the application program must know the length of the work unit currently being processed. When the variable-length or undefined-length format has been specified, TCAM will count the number of bytes in the incoming work unit, add the number of bytes that must be reserved for optional fields in the work area, and place the total either in a special field in the work area or in a field in the input DCB (depending upon which format you specify on the RECFM parameter of the input DCB macro). The application program may then refer to this field to determine the length of the work unit currently being read in.

The length of any output messages or records using the variable or undefined-length must also be specified before these work units can be transferred to TCAM. You, as the application programmer, must ensure that the sum of the work-unit length and the length of any optional fields in the work area has been placed in a prefix field in the work area (variable) or in the output DCB (undefined) before issuing a PUT or WRITE macro to transfer the work unit (see Figure 3-1). (See the following discussion on "Defining Optional Fields in the Work Area" for details on how to specify work unit sizes (see also Figure 3-2).

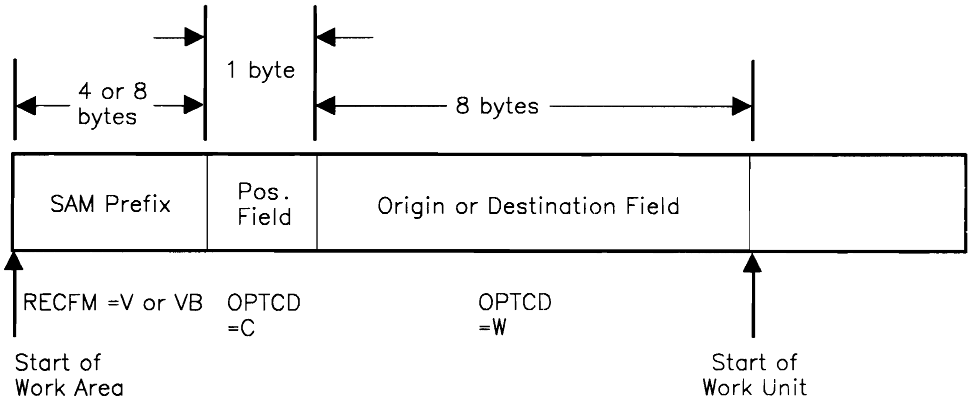


**Figure 3-2. Relative Positions of Optional Fields in the Work Area**

You must specify the work unit format that you wish your application program to accept by coding an appropriate value on the RECFM operand of the input DCB macro. The only values you may code are RECFM=F, RECFM=V, RECFM=U, or RECFM=VB. If you code RECFM=F, TCAM knows that this program is set up to process fixed-length work units and will process only those units that are the same length as specified on the LRECL operand of the input DCB macro.

If you code RECFM=V, TCAM keeps track of the length of each incoming work unit and stores this value in the SAM-prefix field in the program work area each time a GET or READ macro is issued. The total length stored by TCAM is the work unit length plus the length of any optional fields in the work area plus the length of the SAM prefix.

If you code RECFM=U, TCAM adds the length of each incoming work unit to the length of any optional fields and stores this total in the LRECL field of the input data control block. This function is performed each time a GET or READ macro is issued.

For work units being transferred out of an application program, the situation is similar. The RECFM operand of the output DCB macro tells TCAM where to look for the length of the work unit being sent back by each PUT or WRITE. If RECFM=F is specified, TCAM looks for the length of the work unit in the LRECL field of the output DCB. If RECFM=V is specified, TCAM looks for the length field in the SAM prefix to the work area. If RECFM=U is specified, TCAM looks for the sum of the work unit plus the length of any optional fields in the LRECL field of the output DCB. If the WRITE macro is used with the *length* operand, that length is used. It is the application program's responsibility to ensure that the correct length has been entered in every field. The technique for modifying a DCB field is described in the *Data Management Services* manual if you intend to specify RECFM=U.

The significance of specifying a particular format for your work units on an input or output DCB can be summarized as follows:

For the Input DCB:

| **Format** | **How Specified** | **Significance** |
|---|---|---|
| Fixed | RECFM=F | All incoming work units (except possibly the last in a message) are the same length. When a GET or READ macro is executed, TCAM brings in the number of bytes specified by the LRECL operand of the input DCB macro. |
| Variable | RECFM=V | Incoming work units will vary in length. When a GET or READ macro is executed, TCAM brings in data until a delimiter or the end of the work area is encountered. It then places the sum of the length of the work unit plus the length of any optional fields plus the SAM prefix length in the SAM-prefix of the work area. |
| Variable Blocked | RECFM=VB | When a GET or READ macro is executed, TCAM places an eight-byte prefix into the work area receiving a work unit from the input data set for which the DCB macro was coded provided MACRF=R was also coded (a four-byte prefix is provided otherwise). The first two bytes of the prefix contain the binary sum of the length of the work unit plus eight bytes (the length of the prefix) in hexadecimal notation. The second two bytes each contain a binary zero. The third two bytes contain a binary number four less than that contained in the first two bytes. The final two bytes each contain a binary zero. |
| Undefined | RECFM=U | Incoming work units will vary in length. When a GET or READ macro is executed, TCAM brings in data until a delimiter or the end of the work area is encountered. It then places the sum of the length of the work unit plus the length of any optional fields in the work area in the LRECL field of the input DCB. |

For the Output DCB:

| **Format** | **How Specified** | **Significance** |
|---|---|---|
| Fixed | RECFM=F | A PUT or WRITE macro referring to this DCB moves the number of bytes specified in the LRECL field of this DCB from the work area to the MCP. TCAM subtracts the length of any optional fields from the number specified in the LRECL field and only transfers the number of bytes specified for the work unit. |

| Format | How Specified | Significance |
|---|---|---|
| Variable | RECFM=V | When a PUT or WRITE macro referring to this DCB is executed, TCAM determines the length of the work unit to be moved by referring to the SAM-prefix preceding the work unit in the work area. TCAM subtracts the length of any optional fields in the work area before transferring the work unit. |
| Variable | RECFM=VB | When a PUT or WRITE macro referring to this DCB is executed, TCAM assumes that the work unit being sent to the output data set for which the DCB macro is coded is preceded by an eight-byte prefix (provided that MACRF=W is also specified; a four-byte prefix is provided otherwise) whose layout is the same as that described above for the eight-byte BSAM-compatible prefix for the input side. This prefix is for BSAM compatibility; work units are treated as if they were blocked, although only one work unit is transferred for each WRITE macro. It is the application programmer's responsibility to see to it that the prefix contains the proper data before a WRITE macro is executed. |
| Undefined | RECFM=U | If a PUT macro referring to this DCB is executed, TCAM determines the length of the work unit to be moved to the MCP by looking in the LRECL field of the DCB. If a WRITE macro with the "S" operand referring to this DCB is executed, TCAM determines the length of the work unit to be moved by looking in the LRECL field of the DCB. In either case, ACF/TCAM subtracts the length of any optional fields in the work area from the value found. If a WRITE macro with a length operand is executed, TCAM uses the length specified in the WRITE macro minus the length of any optional fields in the work area as the length of the work unit to be moved. |

Figure 3-1 summarizes the specifications and characteristics of the various work-unit formats and types.

The delimiter mentioned in Figure 3-1 in the discussions of variable and undefined-length records is the end of the message when message processing is specified; when record processing is specified, the delimiter may be either the end of the message or a special record-delimiting character that was specified by the RECDEL operand of the TPROCESS macro that created the queue addressed by the GET or READ macro.

If all incoming and outgoing messages were the same length, it would be a simple matter for TCAM to provide buffers of just that size for every application program that processes messages. Unfortunately, this is rarely the case. Instead, messages are usually either too short or too long for a given buffer. Therefore, some sort of compromise buffering scheme must be used. The scheme that is used is the responsibility of the TCAM system programmer, not the application programmer. You, as the application programmer, however, should be aware of how large the TCAM message buffers are. If a buffer is too small for a record, additional buffers must be allocated by TCAM. This may result in inefficient operation or, in some cases, an error. An application program can execute successfully without being aware of exactly what size the TCAM message transfer buffers are; but in the interest of efficient operation, there should be some coordination between the TCAM system programmer and the application programmer concerning the size of TCAM buffers. Therefore, Appendix A provides a more detailed discussion about how buffers are defined in the MCP.

In summary, the length, in bytes, of a fixed-length work unit is specified in the LRECL operand of a DCB macro. The length, in bytes, of a variable-length work unit is defined in the optional SAM-prefix field in the application program work area. (See the discussion on the SAM-prefix field following). The length, in bytes, of an undefined-format work unit is specified in the LRECL operand field of an input or output data control block or in the length field of a WRITE macro. TCAM counts the number of bytes in an incoming work unit and passes this value to the program for both variable- and undefined-length work units. The only difference between the two types of work units (other than the lengths of their respective prefixes) is the location of the field in which TCAM stores the count. For outgoing work units, the same locations are used, and the application program is responsible for entering the appropriate value (see Figures 3-1 and 3-2).

## The Application Program Work Area

Work units obtained by a GET or READ macro are transferred from TCAM to a *work area* defined in the application program. These work areas are similar to those specified in programs using either the basic or queued sequential access methods.

A work area in a TCAM application program may be defined in one of two ways. It may be defined in the application program by a DS or DC instruction. This is a static work area definition. It may also be defined dynamically. Dynamic definition relieves the programmer of having to set aside a specific area in his program for input and output. Every time your program issues a GET or PUT macro, the operating system dynamically acquires a work area. Data movement to or from a static work area always occurs in move mode. Data movement from a dynamically acquired work area must occur in locate mode. The data movement mode is specified on the MACRF operand of the DCB macro associated with the respective logical input and output data sets. See to the *Data Management Macro Instructions* for a discussion on move mode and locate mode.

### Defining a Static Work Area

The label of the DS or DC instruction in your program that defines your data work area becomes the name that is coded in the GET, PUT, READ, or WRITE instruction that moves data to and from a static work area. The size of the work area must be specified in the BLKSIZE operand of the DCB macro associated with the data set whose contents are being transferred to or from the work area.

The APWAS operand of the INTRO macro specifies the size of the work area in the application interface block (AIB). The work area is used as an intermediate buffer area for data transfer between the TCAM address space and the application program address space.

The BLKSIZE operand of the DCB macro specifies the size of the *application program's buffer work area* (in the application program itself) provided by the application programmer.

If the BLKSIZE operand value is greater than the APWAS operand value, the error will be indicated by the appropriate return code that results from GET, PUT, READ, WRITE, or CHECK macro execution.

When a work area is defined in this way, you should specify *move mode* in the DCB macro referred to by the data-transfer macros that use the work area. A static work area may receive data from or send data to more than one input or output data set. Note that either QSAM-compatible or BSAM-compatible request macros may be used for move mode.

## Defining a Dynamic Work Area

If you specify locate mode by coding MACRF=GL on the input DCB macro, execution of the first GET macro referring to the opened data set causes TCAM to dynamically obtain a work area (by a GETMAIN macro) in the same address space as the application program, and to move a work unit of data into this work area.

The APWAS operand of the INTRO macro specifies the size of the work area in the application interface block (AIB). The work area is used as an intermediate buffer area for data transfer between the TCAM address space and the application program address space.

The BLKSIZE operand of the DCB macro specifies the size of the *application program's buffer in the* dynamically obtained work area.

If the BLKSIZE operand value is greater than the APWAS operand value, the error will be indicated by the appropriate return code that results from GET, PUT, READ, WRITE, or CHECK macro execution.

The work-area address is returned to the application program in register 1 and is also saved by TCAM. The second and subsequent executions of the GET macro referring to the same DCB will then move data into this same work area.

If locate mode is specified in an output DCB macro (MACRF=PL), execution of the first PUT macro causes TCAM to dynamically obtain a work area (by a GETMAIN macro) in the same address space as the application program. The length of this work area is specified by the BLKSIZE operand of the output DCB referred to by the PUT macro. The address of this work area is returned in register 1. This address must be saved by the application program so that subsequent output work units may be moved into the same output work area. TCAM will use this same work area until your program ends. No data transfer takes place upon execution of the first PUT macro. The second and subsequent PUTs will move data from the work area addressed by register 1.

*Note:* Only GET and PUT macros may be used with locate mode.

## Defining Optional Fields in the Work Area

The work area can be made up entirely of a work unit. If your application program requires it, however, up to three optional fields can be affixed to the beginning of the work unit. The optional fields are the origin or destination field, the position field, and the SAM-prefix field. One, all, or any combination of two of these fields may be specified depending on your requirements (see Figure 3-2).

These optional fields allow TCAM to pass data concerning the source, position, and size of the current work unit being transferred to the application program on input. On output, your program can fill in these same fields so that TCAM can process the output more efficiently.

The optional fields are specified through operands on the appropriate input and output DCB macros. These operands are:

- OPTCD=[W][C]
- RECFM=V

If OPTCD=C or W is not coded and RECFM is not coded V or VB, TCAM starts with the first byte of the work area. When a work unit is passed to the work area by a GET or READ, it is preceded by whatever optional fields have been specified in the associated DCB. Therefore, you must allow for these fields when defining your work area.

The contents of these optional fields are not moved out of the work area with the message or record being processed by a PUT or WRITE macro. TCAM refers to each field, but does not move it.

### The Origin or Destination Field

If W is coded as one of the parameters of the OPTCD operand of the DCB macro of the input data set, eight bytes at the beginning of the work area are reserved for the name of the source of every message. This eight-byte origin/destination field must immediately precede the work unit in the work area and follow the other two optional fields if they were specified. (See Figure 3-2).

When the first work unit of a message is read into the program work area, TCAM places the EBCDIC name of the source (as specified in the MCP terminal table) into these eight bytes. The name is left-adjusted, and the field is padded on the right with blanks if the name is shorter than eight bytes. This value will remain in the origin field until it is overlaid or deleted. In most cases, it will be overlaid by the source name of the next message.

If TCAM cannot determine the origin of a message, the field is filled with eight character blanks (X‘40’).

If W is coded on the OPTCD operand of the DCB macro of the output data set for this work unit, TCAM looks at the 8-byte field immediately preceding the work unit for the name of the destination of the message when a PUT or WRITE macro is issued to move a work unit from this work area to the MCP. The name should be specified in EBCDIC, left-justified, and padded to the right with blanks if necessary. If a FORWARD macro with the DEST operand coded DEST=PUT is executed by the inheader subgroup of the MH for an application program, the message is sent to the destination specified in the 8-byte field (see the FORWARD macro in the *TCAM Installation Reference* manual). If the destination name is filled with blanks or hexadecimal 0's, the work unit will be passed to the MH with a zero destination and will cause a FORWARD error with DEST=PUT. If you do not route based on the 8-byte prefix set by TCAM or the FHP, then code the destination external LU name in the message just as is done for any external LU, and a FORWARD macro must be coded in the MH.

Only the work unit (without the destination field) is transferred to TCAM when a PUT or WRITE macro is executed. However, the destination field remains a part of the work area as long as OPTCD=W is specified for this DCB.

An inquiry-reply application program whose MH has issued a LOCK macro must have OPTCD=W coded on its input and output DCB macros. TCAM places the message origin in the eight-byte field when the inquiry is initially read into the work area. After the application program processes the message data (without changing the contents of the eight byte field), a PUT or WRITE macro is issued; the contents of the eight-byte field are now assumed to specify a destination address rather than a source address.

For OPTCD=W to work in a TCAM system LUs that can access the application program must be defined to the local MCP using TERMINAL macros. In a large network, this requirement can become a maintenance problem, since the addition of an LU to the network might require the changing of all other MCPs/VTAMs in the network in order to provide communication to the new LU.

This problem may be avoided in an extended network. Routing in an extended network is based on a TCAM network address, which is resolved into a terminal-table entry in the host controlling the message's destination. The network address of the origin and destination of a message flowing in an extended network are contained in a field of the fixed header prefix (FHP) located at the beginning of the message. The TCAM network address of the message origin is contained in an FHP field named FHPOAFLD, while that of the destination is contained in a field called FHPDAFLD. The system programmer can set up the MCP so that the FHP is passed to the application program input work area. The DKJFHP macro allows the user code in the application program to symbolically examine the contents of the FHP.

The application program need not alter the contents of the FHP; code in the incoming group of the AMH may be used to switch the origin and destination TCAM network addresses and route the message back to the origin using the DAFROUTE macro.

*Note:* An option, (controlled by use of KEYDEF macro and MHs in the MCP), allows the 8 character terminal name from the TCAM tables that define originating external LUs to be inserted in the message for application programs that need this information. If this option is used, the name will appear in the message text area.

This technique of routing in an extended network based on the TCAM network address eliminates the need to represent via a terminal macro, each LU that may send messages to these application programs.

For more information on this technique, see *TCAM Installation Guide.* For information on how to access fields in the FHP, see "Inspecting and Changing TCAM Control Elements" in Chapter 4.

The model MCPs contain application program MHs which illustrate ways in which an application program can route messages using a TCAM network address.

*Note:* These examples are particularly adaptable to inquiry-response type programs. Essentially, it is the functional characteristics of the application program which dictate the most appropriate mechanism for routing.

## The Position Field

If C is coded as one of the parameters of the OPTCD operand of an input or output DCB macro, a one-byte field is reserved in the work area in the position noted in Figure 3-2. This field is necessary if messages being sent to the application program are expected to be larger than the application-program work area that is to receive them (for example, when logical records or message segments rather than entire messages, are to be processed by a single GET or READ). On input, TCAM will store a code in this field indicating whether the work unit being passed is the first portion of a message, an intermediate portion, the last portion, or an entire message. (These codes are described in the discussion of the OPTCD operand of the input and output DCB macros in Chapter 5.)

If C is specified on the OPTCD operand of an output DCB macro, the application program must ensure that the position field contains the appropriate code to describe the current work unit for each PUT or WRITE. TCAM checks this field and uses it to account for message segments being transferred to the MCP. You must not interleave segments for different messages. TCAM has no way of determining which segment belongs to which message. If C is not specified on the OPTCD operand of an output DCB macro, TCAM will make one of two assumptions, depending upon whether record processing or message processing is specified.

- If message processing is specified (OPTCD=U), the end of the work unit is assumed to be the end of the message. That is, TCAM assumes that one work unit equals one message.

- If record processing is specified or assumed, TCAM assumes that all work units specifying this output DCB macro being sent to the MCP from the time the output data set is opened until the time it is closed are part of the same message. That is, the application program signals end-of-message by issuing a CLOSE macro after the last work unit in the message is sent to the MCP.

The position field is located in the work area immediately preceding the eight-byte origin/destination field. If no origin/destination field is present, this field will immediately precede the first byte of the work unit.

## The SAM-Prefix Field

If RECFM=V or VB are coded on an input or output DCB macro, a four- or eight-byte SAM-prefix field must be present in the work area. TCAM uses this field to insert or refer to the length of the associated work unit.

If RECFM=V is coded on an input DCB macro, TCAM places four bytes into the SAM-prefix field of the work area receiving a work unit. The first two bytes of the prefix contain the binary sum of the length, in bytes, of the work unit plus four (the length of the prefix). The second two bytes of the prefix contains binary zeros.

If RECFM=VB and MACRF=R are coded on an input DCB macro, TCAM places eight bytes into the SAM-prefix field of the work area that receives a work unit. (If RECFM=VB is specified and MACRF=G is specified, only four bytes are provided.)

The first two bytes of the prefix contain the binary sum of the length, in bytes, of the work unit plus eight (the length of the prefix) plus the length of any other optional fields. The second two bytes contain binary zeros. The third two bytes contain a binary number that is four less than the number contained in the first two bytes. The final two bytes contain binary zeros. This eight-byte prefix is for BSAM compatibility; work units are treated as if they were blocked records, although only one work unit is transferred for each READ or GET macro executed. You are restricted to a blocking factor of one for RECFM=VB.

If RECFM=V is coded on the output DCB macro, TCAM assumes that a four-byte SAM-prefix precedes each work unit being sent. This prefix is similar to a standard SAM variable-length prefix. Its contents are described in the discussion of the SAM-prefix for input data sets. It is the application program's responsibility to see that the prefix contains the proper data before a PUT or a WRITE macro is issued.

If RECFM=VB and MACRF=W are coded on an output DCB macro, TCAM assumes that the work unit being sent is preceded by an eight-byte SAM-prefix; a four-byte prefix is expected otherwise. The layout of these eight bytes is the same as described for the eight-byte BSAM-compatible prefix for input data sets. This prefix is also for BSAM compatibility; work units are treated as if they were blocked, although only one work unit is transferred for each WRITE macro. It is the application program's responsibility to see that the work unit and all its prefixes contain the proper data before a WRITE macro is executed. You are restricted to a blocking factor of 1 for RECFM=VB.

The origin/destination field, the position field, and the SAM-prefix field are the only fields that can be specified in the option field portion of the work area. The remainder of the work area is made up of a work unit. The options for these fields may be included at program execution through a DD card parameter (the DCB OPTCD operand) or at assembly through the appropriate DCB macro operands. Figure 3-3 is a cross-reference matrix that shows how to code each OPTCD operand parameter.

| Operand | Use | Relative Position | Length of Field | Notes |
|---|---|---|---|---|
| RECFM= V or VB, or U | SAM prefix | 1 | 4 bytes or 8 bytes | If V or VB, this field contains the length of the current work unit; TCAM fills this field on input; the user must fill this field on output. If U, the work unit is a message or segment that is not a record. |
| OPTCD= C | Position | 2 | 1 byte | The indication in this field specifies whether the segment currently being processed is the first, an intermediate, or the last segment. |
| OPTCT=W | Origin/destination field | 3 | 8 bytes | TCAM loads name of message source here on input. On output, the user must enter message destination. |

**Figure 3-3. OPTCD Coding Matrix**

## Message Handling Considerations

Regardless of how large or small a message is, it usually consists of two parts—the header portion and the text portion. However, a message may be either header only or text only. The header portion contains control information relating to that message, such as:

- One or more message destination codes
- The name for the originating external LU
- The number of the message relative to the numbers of the previous messages received from that external LU (input sequence number)
- A message-type indicator
- Several fields containing TCAM control indicators.

The text portion of a message usually consists of information that is of concern only to the destination.

A header-only message may use some sort of user-created message-type indicator to route the message to an application program and, possibly, obtain a standard reply. If all messages of a particular type go to only one application program, such as a file-update program, the header may be omitted. The determination of what part of a message shall be processed as a header and what part as text is up to the TCAM system programmer.

Operations on the fields in an MH is primarily the responsibility of the TCAM MHs. A MH can, however, pass on some or all of the header to an application program if the application so requires. The length and format of the header and the information in it are the responsibility of the user and are specified in the MCP by the TCAM system programmer. By the same token, message control information may be included as part of the text. For example, you may code your own control block or header as a part of the work unit.

You should work closely with the TCAM system programmer whenever you design an application in which more than the basic TCAM control information referred to in Chapter 2 is to be exchanged between TCAM and an application program.

Several MH macros used in the MCP can be coded in the application MH. However, the following macros cannot be coded in the application MH: CANCELMG (in the outheader), LOCK, MSGFORM, MSGGEN, SCREEN, IEDDFC, IEDHALT, IEDRESP, IEDRH, IEDSENSE SETEOM, and UNLOCK. See the *TCAM Installation Guide* for a discussion on the macros that can be coded in application MHs.

Application MHs can use the TCAM capability of routing by key by using the KEYPROC macro to route messages. Routing by key allows you to take advantage of the following capabilities:

- Special management capability for extended interchanges with a single application program (end-to-end session)
- Security capability, which allows only authorized message originators to interact with a particular application program
- Notification capability to tell a message originator that the application program with which he wishes to communicate is not currently active.

If you use the TCAM capability of routing by key, you can use the code in the model MCP as an example in setting up your own MCP. For more information on routing by key, see the sections "TCAM Overview" and "Coding the Message Handler" in the *TCAM Installation Guide*.

## Specifying an End-of-Message or End-of-File Routine

TCAM provides a capability for signaling the application program that the work unit currently being processed is the end of a logical file of data; after processing this work unit, the application program will take an exit to a user-defined end-of-data routine. Such a routine could close the input data set, cause a different type of application-program activity to begin, or issue a GET or READ macro referring to a different PROCESS queue. Again, the decision concerning how you use this function depends on your application. To use this function, you must first specify an address label as the EODAD operand of your input DCB macro. This address must specify a routine that you have coded to handle the end-of-data circumstance.

When TCAM recognizes an end-of-file condition on an input data set, it can indicate that the contents of the current work unit represent the final portion of a logical file of data by executing a SETEOF macro in the application-program MH. The SETEOF macro can be coded to execute conditionally based on the presence of a specified character string (such as an end-of-file indicator) in the message header. When a SETEOF macro executes in the MCP, a bit is set in the TCAM buffer prefix of the message, indicating that this is the last message in the data set. When a message with this bit on in the prefix is transferred to the application program by a GET or READ macro, TCAM notes that this is the last message in the file.

Execution of the first READ, GET, or CHECK macro following transfer of the entire end-of-file message to the application-program work area gives control to the routine specified by the EODAD operand of the related DCB macro.

Upon recognition of the end-of-file condition and subsequent execution of a READ, GET, or CHECK macro, TCAM loads register 15 with the address of the exit routine and control transfers to the EODAD routine. Upon returning from the EODAD routine to the application program issuing GET or READ macros, GET or READ macros will again execute in a normal fashion.

If no SETEOF macro has been specified in the MCP and a GET macro referring to an empty input queue is issued, the application program enters a wait state until a message subsequently arrives at the read-ahead queue for that application program. If a READ macro was issued, the wait state begins only when the check macro is executed.

If the SETEOF macro executes and no EODAD exit was specified, a completion code is returned to the program in register 15 and control is returned to the application program when the next READ, GET, or CHECK macro is executed. Your program should then check for this return code and take appropriate action.

If record processing is specified in your program by the absence of the OPTCD=U parameter on an output DCB macro, you may indicate to the MCP that this is the last record in a message by coding X'F2' in the position field preceding the record in the work area. (See "Defining Optional Fields in the Work Area" in this chapter for a description of the position field.) If no position field is defined, the program can signal the MCP that the last record in the message has been sent by closing the output data set after executing a PUT or WRITE macro for this last record, or by issuing a PUT or WRITE macro with no data to be moved. A PUT or WRITE macro for a header segment is ignored as an invalid operation when no data is in the work area. (If message processing is specified and no position field is provided, TCAM assumes that the work unit being processed is an entire message).

In summary, if the EODAD operand is specified on an input DCB macro, the SETEOF macro may be issued in the MCP to indicate the end of a file of data, and the EODAD exit is taken on the next GET or READ macro after TCAM moves the end-of-message into the program work area. On a succeeding GET or READ macro, normal processing continues. If EODAD is not specified, the application program is responsible for issuing a CLOSE macro to close the input DCB.

If no SETEOF macro had been specified in the MCP, each GET or READ with CHECK will not be completed until a message arrives on the TCAM input queue. End-of-file processing must be handled by some means other than using the EODAD exit. By using the SETEOF macro function, time of entry to EODAD can be controlled by the user.

### Summary of Related Operands

Values for the following TCAM macros must be coded before the SETEOF macro function will execute properly.

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| SETEOF | | Application MH in the MCP | The operands will specify how end-of-file is recognized by TCAM. See the *TCAM Base Installation Reference* |
| DCB (input) | EODAD | Application program | |

## The Data Transfer Macros

Although messages in the TCAM environment are usually received from (or sent to) external LUs or SNA LUs over communication links, you will be able to use the same form of data-transfer macros that are used in local environments.

A TCAM message control program performs all input/output operations for an application program. Although TCAM can be described as a queued access method, it provides either a basic access technique or a queued access technique for I/O processing at the application program interface. The technique that you choose will depend on your application program requirements.

### The Basic Access Technique

The basic access technique uses the READ, WRITE, and CHECK macros for passing data between the TCAM MCP and an application program. The basic technique should be used when you cannot predict the sequence or timing of message arrival, or if you do not want all the automatic functions performed by the queued access technique. (See the "Queued Access Technique" discussion following). Buffers allocated either by you or the operating system are loaded or emptied individually each time a READ or WRITE macro is issued. Issuing a READ or WRITE macro only initiates the I/O operation. To ensure that the operation has completed successfully, you must issue a CHECK macro or another READ macro to test the data event control block (DECB) of the DCB controlling the I/O you are performing. Optionally, you can test the DECB completion code yourself instead of issuing a CHECK macro.

Since the TCAM support for the READ, WRITE, and CHECK macros is similar to that provided by MVS, you are expected to be thoroughly familiar with the MVS basic sequential access method (BSAM). This requirement implies a knowledge on your part of the applicable contents of *Data Management Services* and *Data Management Macro Instructions.*

## The READ Macro

The READ macro instruction requests TCAM to transfer a work unit from the MCP to a designated work area in the application program. After a READ macro is issued, control may be returned to the application program before the work unit is actually transferred. Therefore, every read operation should be tested for completion by issuing a CHECK macro. Once a CHECK macro is issued, control is not returned to the application program until a new work unit has been placed in the work area. Input is used here as meaning input to an application program from the MCP.

The *decbname* and *dcbname* operands of a READ macro, once specified, should always be paired; *decbname* should not be specified with a particular *dcbname*in one READ macro and then associated with a different *dcbname* in another READ macro in the same program. You may specify only one DECB per DCB. This technique allows the user to determine the status of any process queue by interrogating the current completion code in the DECB. (See "Completion Codes for the Basic Access Technique" on page 3-25.)

Since only one DECB may be specified per input queue, multiple READ macros directed to the same input DCB are not permitted. However, you may achieve the effect of issuing multiple READ macros directed to the same DCB by coding a list and execute form for the same READ macro. You must code at least one list form of the READ macro and then you may code several associated execute-form macros. The list-form READ macro and all the execute-form READ macros would specify the same DCB and DECB. This technique does not actually provide a multiple-wait facility, but does allow you to code READ macros that refer to the same DECB in one or more sections of the same program. (Details of the function of the list and execute forms of the READ macro are explained in *Data Management Macro Instructions*.)

For instance, in the following example, two READ macros of the execute form and one READ macro of the list form are coded. All three macros specify the same DECB (named INPUT). The list-form READ macro also specifies the appropriate DCB and work area.

```
                          User Code

             READ         INPUT,SF,MF=(E,LIST)
             CHECK        INPUT
                          ●
                          User Code
                          ●
             READ         INPUT,SF,MF=(E,LIST)
             CHECK        INPUT
                          ●
                          User Code
                          ●
                          Constant Area
LIST         READ         INPUT,SF,INDCB,INAREA,MF=L
INAREA       DC           50F'0'
INDCB        DCB          DSORG=PS,MACRF=R,BLKSIZE=200,*
                          OPTCD=WUC,RECFM=V,DDNAME=IN
```

In the TCAM environment, the DECB is posted when a message is placed on a previously empty Read-Ahead queue for a TPROCESS entry. This means that a DECB may be posted any time during the execution of an application program after the first READ macro is issued following the execution of an OPEN macro. This differs from BSAM in that a DECB becomes eligible for posting only after a READ macro is issued.

Therefore, under TCAM, a READ DECB may already be posted complete when you issue a READ, CHECK, or WAIT macro. You should design your program accordingly.

If a READ macro is issued after a quick closedown of the TCAM message control program has begun, the EODAD exit specified on the input DCB macro is taken if the STOP operand is also coded in that same DCB macro. An EODAD exit may also be taken if a SETEOF macro was executed in the MCP as described previously.

## The WRITE Macro

The WRITE macro instruction moves the current contents of a work area in your application program to a buffer in the MCP. As with the READ macro, it is possible that control may be returned to the application program after the WRITE macro is issued, but before the contents of the work area are actually moved. Therefore, this output operation should also be tested for completion by using a CHECK macro instruction.

WRITE macros may also be specified more than once in any application program. The destination of outgoing work units may be specified either in the destination field in the program work area, in the message header in the data portion of the message, or by an operand of the FORWARD macro in the MCP.

If a WRITE macro is issued after a quick closedown of the MCP has begun, the write operation does not complete, and a completion code is placed in the DECB associated with the WRITE macro. (See the discussion on BSAM completion codes following.)

## The CHECK Macro

The CHECK macro instruction causes an application program to be placed in the wait state until an associated input or output operation initiated by a READ or WRITE macro is completed. The DECB controlling the input or output operation is tested by the CHECK macro for an error indication. If no error occurred, control returns to the application program at the instruction following the CHECK macro. If an error occurred, the routine specified by the SYNAD operand of the associated DCB macro is given control. If no SYNAD routine was specified and an error occurs, a return code of X'08' is passed to the program in register 15 after the CHECK macro has executed. The program is then responsible for recognizing this code and the subsequent error processing.

Whenever an end-of-data condition is recognized after a CHECK macro was issued, CHECK passes control to a user-specified EODAD routine. If you do not specify an EODAD routine, your program is responsible for checking the associated event control block (ECB) for a completion code and then performing appropriate end-of-data processing itself.

A CHECK macro should be issued after each READ or WRITE macro in the same order as the READ or WRITE macro instructions were issued. If data is immediately available when a READ macro is issued, it is moved into the program work area, and the event control block (ECB) in the data event control block (DECB) for that READ macro is posted with a completion code. The CHECK macro verifies this operation and immediately returns control to the application program.

*Note:* You do not have to check these codes in the DECB yourself; the CHECK macro accomplishes this for you.

If data is not immediately available and a user-specified end-of-file has not been recognized by TCAM, the application program goes into a wait state while the CHECK macro waits for data to arrive and the ECB to be posted. When data becomes available, TCAM causes it to be moved into the work area. When this has been accomplished, the application program again receives control.

If you choose, you may issue a WAIT macro instead of a CHECK macro to provide a multiple-wait capability. This is accomplished by specifying the ECB address in the ECB operand of a WAIT macro. (The ECB is contained within the first four bytes of the DECB and is located on a fullword boundary.) The function of the WAIT macro is described in *Data Management Macro Instructions*.

You may also wish to write your own code to detect an empty-queue completion code in the DECB. By using this alternative, your program can do other processing while waiting for I/O to complete. After you complete this additional processing, you may check the ECB completion code to see if it has been altered due to a message being placed on the associated read-ahead queue. You must then issue either a CHECK macro or another READ macro to cause the pending READ to complete. This technique requires one DECB per input queue.

When the DECB address is made available to the MCP at the time of the first read operation, the MCP posts that DECB when a message is queued on the previously empty queue in the MCP. It is possible that the DECB may be posted complete after a CHECK macro is issued, but before the next READ macro is issued. You should be aware of this and be able to handle this possibility in case of a multiple wait. The DCB must be closed and reopened to continue processing

## Completion Codes for the Basic Access Technique

After you have issued a READ or WRITE macro, and the BSAM data movement routine has completed execution, a completion code is placed in the ECB field of the data event control block associated with the respective READ or WRITE macro. These codes appear in the data event control block, not in a general purpose register, and are examined by the CHECK macro. The CHECK macro inspects the DECB and transfers control according to the code issued.

If you prefer to issue a WAIT macro rather than a CHECK macro, you are responsible for testing the completion code yourself. The completion codes are stored in the high-order byte of the first word of the DECB. The codes and their meanings are described in the CHECK macro discussion in Chapter 5.

To aid you in coding, you may use the chart in Figure 3-4.

For the READ Macro

| ECB Code | Code Meaning-READ | Next Action | A.P. Wait | Next Code |
|---|---|---|---|---|
| 7F | Normal completion—Data in work area | READ | No | -- |
| 70 | Either SETEOF executed in MCP (work unit not in work area) or TCAM closedown in progress | CHECK<br>CLOSE | No | 70 |
| 52 | Work-area overflow | CHECK<br>CLOSE | No | 52 |
| 50 | Message not found or message incomplete on queue after POINT | READ<br>CHECK<br>CHECK<br>CLOSE | Yes<br>Yes<br>Yes | 7F<br>7F<br>50 |

**Figure 3-4 (Part 1 of 2). Completion Code Matrix**

For the WRITE Macro

| ECB Code | Code Meaning-WRITE | Next Action | A.P. Wait | Next Code |
|---|---|---|---|---|
| 40 | Data on read-ahead queue | READ | Yes | 7F |
| | | CHECK | Yes | 7F |
| | | Continue other A.P. processing | No | |
| 02 | No data available | CHECK | Yes | 7F |
| | | WAIT | Yes | 40 |
| | | Continue other A.P. processing | No | |
| 01 | No data immediately available but some data currently resides on destination queue | CHECK | Yes | 7F |
| | | WAIT | Yes | 40 |
| | | Continue other A.P. processing | No | 40 |
| 7F | Normal completion | WRITE | No | |
| 5E | Request for quick closedown in effect | CLOSE | | |
| | | CHECK | No | 5E |
| 5C | Destination message queues data set congested with traffic | WRITE | No | Any |
| | | Continue other A.P. processing | No | |
| | | CHECK | No | 5C |
| 58 | Work unit sequence error | CHECK | No | 58 |
| | | CLOSE or correct error and issue WRITE | No | Any |
| 54 | Invalid message destination | CHECK | No | 54 |
| | | CLOSE or correct error and issue WRITE | No | Any |
| 52 | Invalid work area size | CHECK | No | 52 |
| | | CLOSE or correct error and issue WRITE | No | Any |

**Figure 3-4 (Part 2 of 2). Completion Code Matrix**

## The Queued Access Technique

The queued access technique uses the GET and PUT macros to pass data between TCAM and an application program.

Because the operating system synchronizes I/O with other application program processing when the queued access technique is used, there is no need to test for I/O completion. After a GET or PUT macro is issued, control is not returned to your program until an input area has been filled, or an output area has been queued for its destination. Exits to error analysis (SYNAD) and end-of-volume or end-of-data (EODAD) routines are taken when required. Move or locate mode may be specified for the queued access technique, but substitute mode is not supported in TCAM.

Since the TCAM support for the GET and PUT macros is similar to that provided by MVS, the TCAM application programmer is expected to be thoroughly familiar with the MVS queued sequential access method (QSAM). This requirement implies a knowledge on your part of the applicable contents of *Data Management Services* and *Data Management Macro Instructions*.

### The GET Macro

The GET macro transfers a single work unit from the TCAM message control program to an application-program work area. The GET macro may be coded more than once in an application program. The size and format of the work unit transferred depends upon the BLKSIZE and LRECL operands, and whether record or message processing is specified by the OPTCD operand of the input DCB macro associated with this GET macro.

If a GET macro follows a POINT macro (see "Retrieving Messages from TCAM Disk Data Sets" in Chapter 3) and the message cannot be retrieved for some reason, a code is returned in register 15 after the POINT macro is executed. See the POINT macro discussion in Chapter 5 for details on the return code. You may want to check for these possibilities before issuing a GET macro if you are retrieving messages.

If a GET macro is issued after a closedown of the MCP has begun, the EODAD exit is taken if the STOP operand is coded on the DCB (see discussion in Chapter 5). If a GET macro is issued and the value of the BLKSIZE operand in the DCB macro is zero, a code is returned in register 15.

### The PUT Macro

The PUT macro causes a work unit to be transferred from your application-program work area to the TCAM message control program, where it is processed and placed on a destination queue. PUT macros may be specified more than once in an application program. The work unit destination may be specified either: (1) in the message header, (2) as an operand of a FORWARD macro in the MCP, or (3) in the destination field in the work area. You are responsible for loading an address in the message header or destination field before the PUT macro is issued.

If your program issues a PUT macro and the MCP cannot immediately accept the message because message traffic in TCAM is currently too heavy, the PUT macro does not execute, a return code indicative of that fact is passed back in register 15, and control passes to the next instruction in the application program. This will occur as a result of the TCAM facility that implements input slowdown because of a temporary maximum message processing threshold condition. You should test the return code after each PUT so that you may reissue the macro at a later time if slowdown has occurred.

If a PUT macro is issued after a quick closedown of the MCP has begun, the operation does not complete, a return code is placed in register 15, and control passes to the next instruction. There is a complete list of return codes in the PUT macro discussion in Chapter 5.

### Caution Against Issuing Simultaneous PUT and WRITE Macros

Do not try to execute a PUT or WRITE macro in a TCAM application program if there is a possibility that another PUT or WRITE macro may be currently executing either in another region or as a subtask, and indirectly referring to (by the TPROCESS entry) the same process control block (PCB). This condition could occur if two subtasks of the same application program with a single PCB tried to execute a PUT or WRITE macro.

To guard against this condition, TCAM returns a warning indication. If an attempt is made to execute a PUT or WRITE macro to a PCB entry that currently has another PUT or WRITE macro pending, TCAM will return a completion code of X'5C000000' in the DECB for a WRITE macro and X '10' in register 15 for a PUT macro. Unlike other PUT or WRITE macro errors, the user is not required to close down the DCB affected in this instance.

If more than one subtask in the same application program includes PUT or WRITE macros, the possibility of this type of error can be eliminated through use of the ENQ and DEQ macros. An ENQ macro should be coded before each PUT or WRITE macro, and a DEQ macro should be coded after each PUT or WRITE macro. The resource specified on the ENQ macro must be the symbolic name of the process control block you wish to control. Using this facility will protect against issuing a simultaneous PUT or WRITE macro to the same PCB. See *Supervisor Services and Macros* for information on how to code an ENQ or DEQ macro. Also, see Appendix I of the *IBM System/370 Principles of Operation* manual for examples of how to use the compare and swap instruction to lock a serially reusable resource.

## Retrieving and Retransmitting TCAM-User Messages

Occasionally during the operation of a network it may become necessary to retrieve and/or transmit a message or series of messages that have already been processed by TCAM. This includes both messages that have been transmitted to their destinations and messages still being held by TCAM for transmission. Either the POINT macro or the QRESET macro will allow you to handle most of the message retrieval or retransmission situations that may arise in your environment.

The online retrieval system service program uses TCAM message retrieval macros to retrieve messages, based on operator commands. For details, see the *TCAM Utilities* publication.

While TCAM is processing or storing a message, that message resides on a message queue data set. Message queue data sets are controlled by TCAM, but in some instances the messages in these data sets can be accessed by an application program. Depending on how your TCAM network was generated, the message queue data set may reside:

- On reusable disk
- On nonreusable disk
- In main storage only
- In some combination of main storage and disk storage

As long as your network uses some form of disk queuing, you will be able to use either or both of the following facilities.

## Retrieving Messages from TCAM Disk Data Sets (the Point Macro)

If your TCAM system does not use disk storage for the TCAM message queue data set or disk backup for a main-storage message queue data set, you cannot use the POINT macro.

TCAM uses a combination of a GET or READ macro instruction and the POINT macro to return a specific message to a work area in your program.

The POINT macro must specify the symbolic label of a particular destination queue or a TPROCESS entry and the message sequence number of the particular message you wish to retrieve. You must also provide a work area in your program to which the message can be returned.

A message may be retrieved even if it has already been sent to its destination, provided that the entire message is still on disk in the message queue data set at the time that the POINT macro is executed.

Since an output sequence number is not assigned to a message until it is actually sent to the destination, no message can be retrieved by *output* sequence number until after it has been successfully sent.

The POINT macro does not provide for nonsequential retrieval of message segments. For example, if a GET macro moves a header segment into the program work area after issuing a POINT macro, and then a second POINT macro is issued, the next GET macro will retrieve the next segment of the current message. This sequence will continue until all subsequent message segments of that particular message have been moved to the program work area. If you do not wish to retrieve the rest of a message, you may specify a POINT macro whose *address* operand specifies a destination name followed by a X‘40’. (See the description of the *address* operand in the POINT macro discussion in Chapter 5).

If your application program work area is too small to contain an entire message, the next GET or READ macro referring to the same DCB retrieves the next segment or the rest of the message (if OPTCD=C is specified). If OPTCD=C is not specified, the SYNAD exit is taken.

## Retrieval of Multiple Messages

By using this function, TCAM allows you to request retrieval of either a single message, or a series of messages. It also allows the return of all information in the queue-back chain for a specific destination. If you request information from a queue-back chain, you are then responsible for examining this information to find the desired message. A queue-back chain is a time-sequenced record of the message traffic (both sent and received messages) for the external LU or external LUs represented by a specific destination queue control block (QCB). TCAM maintains this chain specifically for the message retrieval function for application programs, as this queue-back chain contains all messages both sent and unsent messages can be retrieved. Only messages that have already been completely sent can be retrieved by output sequence number. For additional information see "queue-Back Chain under Nonreusable Disk Queuing" in Section 1 of *TCAM Diagnosis Reference.*

The advantage of using the multiple retrieval function is that it reduces the amount of disk access activity when you want to retrieve more than one message.

To perform multiple retrieval, the application program must set up the required input for the *address* operand of the POINT macro, set the high-order bit in the sequence number field in the POINT macro work area (see the POINT macro discussion in Chapter 5), and then issue the POINT macro followed by a GET or READ macro for each message.

Multiple retrieval by sequence number will retrieve the message with the sequence number specified and every message on that queue with a *lower* sequence number. When retrieving by output sequence number, unsent messages will be returned if the character O was coded at X'8' into the POINT parameter list. If S (MVS only) was coded instead, then only the sent (serviced) messages will be retrieved. All messages whether sent or unsent are returned in the order in which they reside on the queue-back chain, that is, they will be returned in the reverse order in which the header buffers were received. Multiple retrievals are terminated when the end of the queue-back chain is reached. This will be indicated by a return code in REG15 of X'0000000C' when the GET is issued.

When retrieving multiple messages, your work area for the POINT macro must be large enough to contain an entire message segment plus ten bytes. This is because the following information is moved into your work area in front of each retrieved message:

| **Field** | **Number of Bytes** |
|---|---|
| Source offset | 2 |
| Message status | 1 |
| Input sequence number | 2 |
| Output sequence number | 2 |
| Destination offset | 2 |
| Number of reserves | 1 |

The contents of each of these fields are:

The *source offset* is the binary value of the index into the TCAM terminal table for the source of this message.

The *message status* is a byte containing the following work unit status information. The Test-Under-Mask instruction may be used for testing this byte. You may ignore any bit combinations not shown here.

| | |
|---|---|
| X'01' | This is not the first buffer of a message. |
| X'02' | This is not the last buffer of a message. |
| X'04' | FHP present. |
| X'08' | This is a duplicate header. |
| X'40' | An error message is in this buffer. |
| X'80' | A CANCELMG macro has been executed. |

The input sequence number provides TCAM a means of ensuring that messages are received from a source in the correct order.

The output sequence number is placed in the header of a message by TCAM. It specifies the order in which messages were sent to a destination by TCAM. You may use this field and the input sequence field for your own verification.

The destination offset is the binary value of the index into the TCAM terminal-name table of the symbolic name of the destination of this message.

The number of reserves specifies the number of reserved bytes that precede the message data in the work unit. For multiple retrieval, the reserve characters are moved to the work area as data. See the RESERVE operand of the Group macro in *TCAM Installation Reference* for a discussion of reserve bytes.

To perform multiple retrievals, an application program must set up the required input to the POINT macro (set the high-order bit in the sequence number field to trigger the function, and then issue the POINT macro followed by a GET or READ macro for each message). See the POINT macro discussion in Chapter 5 for details on how to code the *address* field.

The code sequence in your program would be:

```
(initialize POINT
address
field)
POINT
READ
(read processing)
POINT
READ
(read processing)
```

Because the POINT parameter list is manipulated by TCAM during the retrieval process, the parameter list should not be altered, following initial setup, until the retrieval function has terminated.

At least one complete message must be fully retrieved before retrieval of another message can be started. If your application program work area is not large enough to contain a complete message, a single GET or READ obtains only a segment of the message. Remember that execution of the POINT macro should precede each issuance of a GET or READ during the retrieval process.

You may terminate the multiple retrieval facility prior to receiving all messages by issuing a POINT macro with a blank following the address field. However, this last POINT macro must still be followed by a GET or READ macro. Retrieve termination must not be invoked by the application program if retrieve runs to its normal end or if TCAM returns a non-zero return code before normal end. This does not apply in the case of return code X'14' from POINT. This return code implies that an attempt was made to start a retrieval when one was in progress. It is true that the new one would not be started; however, the original must be ended before anything else may be done. Retrieve termination may also be accomplished by closing and reopening the DCB.

## Summary of Related Operands

Values must be specified for the following macro operands before the POINT macro will execute properly.

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| DCB. | OPTCD=C<br>MACRF | Application Program<br>GLT<br>GMT<br>RP | If a message is expected to be larger than the program work area, this parameter must be specified. |
| INTRO | DISK=YES | TCAM MCP | If DISK=NO is specified, POINT cannot execute. |

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| GROUP | RESERVE | TAM MCP | Specifies the number of reserved bytes preceding a work unit that contains ACF/TCAM control information. This value does not refer to the option fields preceding the work unit. |
| TERMINAL or TPROCESS | QUEUES | TCAM MCP | Must have disk queues specified if messages are to be retrieved. |

## Retransmitting Messages from the TCAM Message Queue Data Set (the QRESET Macro)

During the operation of your TCAM network, you may occasionally wish to resend a message that has already been transmitted to an output device or logical unit (LU) and marked serviced by TCAM. A message is marked as serviced when it has been completely transmitted to its destination; a TCAM routine has set an indicator in the message queue data set that marks the copy of the message on disk as serviced. For a message transmitted to an external LU or LU, the service indicator is set when acknowledgment is received that the entire message has been successfully received. For a message sent to an application program, the service indicator is not set until the next sequential message has been entirely moved into the application-program work area.

A temporary hardware problem such as a broken printer ribbon at an external LU may precipitate a situation where you would want to retransmit a particular message. QRESET allows you to request resumption of an output operation from a specific message queue at a particular output message sequence number. Only messages that have been placed on a destination queue on a disk data set can be retransmitted.

When you issue the QRESET macro for a particular message the specified queue is searched for the sequence number specified. As the queue search proceeds backward from the last message sent to the specific external LU, all intervening messages, up to and including the requested one, are marked unserviced. This process is called "resetting the queue". The necessary control fields in TCAM are also reset to allow resending all the marked messages.

The output-sequence-number field in the associated terminal-table entry is replaced with the lowest output-sequence-number of all the messages reset. All messages with higher sequence numbers were also reset and are available in the same sequence as the original. This ensures output-sequence-number integrity for that queue when a reset message is retransmitted. This ensures output-sequence-number integrity for that queue when a reset message is retransmitted.

"Good morning" messages cannot be processed by QRESET. Neither initiate mode nor lock mode messages will be processed for resending. If a message with the requested sequence number or any intervening numbers cannot be marked unserviced, the count of messages that have been successfully processed by QRESET is returned in register 2 and an appropriate return code is provided in register 15. This count includes only those messages with output-sequence numbers higher than the first initiate mode or lock message encountered. (See the QRESET macro discussion in Chapter 5.)

You should be aware that while a queue is being processed by the QRESET macro, the application program that issued the QRESET macro is put into a wait state, and any output from other sources to the external LU whose queue is being processed is temporarily suspended. Error conditions that terminate processing any queue reset request are listed in the discussion of the QRESET macro in Chapter 5. There is one important consideration when using the queue reset facility for any device whose destination message queue resides on reusable disk. In searching the message queue for the requested output sequence number, only header portions of messages are examined. It is possible that an intact header resides on disk but that its text segment has been overlaid as a result of reusable disk reorganization. If this situation exists and TCAM tries to resend a partially destroyed message, a logical read error will occur and TCAM will terminate abnormally. Therefore, if the QRESET macro finds that a header lies within a disk storage zone that has been or is about to be reorganized, the queue reset function terminates with an appropriate return code. In addition, no attempt is made to resend any of the messages that may have been successfully processed by the QRESET macro up to this point. See the message data sets discussion in *TCAM Installation Guide* for details on disk storage zones.

If a header unit may be safely retransmitted, the QRESET macro checks the position of the first text unit of the message relative to the disk location of the header unit. If it is determined that the text unit is far enough behind the header to present problems when resending, the QRESET macro does not mark the message unserviced. However, processing of remaining requested messages continues.

To use queue reset with the least impact on the issuing application program, on the external LU whose queue is being searched, and on overall processing time, you should resend only small numbers of recently transmitted messages.

In addition to the requirement that the queue reside on disk:

- There can be no priority level queuing on the queue you want to reset.
- The external LU whose queue is being reset must not be intercepted (held) at the time that the QRESET macro is issued.
- If the queue of an application program is being reset, the DCB in the application program for this queue must be closed.

The QRESET macro is coded in an application program. The TPROCESS macro that points to this application program must have specified QBACK=YES. Before you can execute the QRESET macro, you must provide TCAM with the desired message sequence number and the name of the output device or the application-program TPROCESS entry whose queue you wish to be reset. The address of a ten-byte field containing this information must be placed in register 2 before the QRESET macro executes. The format of this field is:

| Byte | Format | Contents |
|---|---|---|
| 0-7 | Character (left-adjusted and padded with blanks) | Destination name: the symbolic name of the external LU or TPROCESS entry for which you wish messages to be reset. |
| 8-9 | Hexadecimal | Sequence number (maximum value is decimal 9999 or X'270F') of the message from which you wish to begin the reset process. All higher sequence-numbered messages will also be reset. They must be less than the current SEQOUT number. |

## Summary of Related Operands

The following operands must be coded before the queue reset facility can be used.

Required Macros and Operands:

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| TPROCESS | QBACK=YES | MCP | This informs TCAM that the application program pointed to by this TPROCESS macro will use the queue reset facility. |

# Chapter 4. Optional TCAM Facilities for the Application Programmer

This chapter describes several TCAM functions available to the application programmer that are not directly related to the transfer of messages. Subjects covered include:

- Coding operator control commands in an application program
- Inspecting and changing TCAM control blocks
- Time stamping and counting messages
- Displaying storage from within an application program
- Converting numbers into different formats

The checkpoint facility is discussed and coordination of system and application program checkpoints are described. This chapter contains programming detail that may not be understood by someone without a programming background.

The TCAM functions that have been discussed up to this point have dealt primarily with defining the interface between a TCAM message control program and an application program, and the passing of messages across that interface. Through the functions discussed in this chapter, you can build more reliability and convenience into your system. An increase in reliability can be obtained through the checkpoint and operator command capabilities, and convenience, through the message counting and dating and control block inspection capabilities. The use of any of these facilities will depend upon the size and function of your TCAM system and your application program environment.

## Issuing TCAM Operator Control Commands from an Application Program

VTAM provides the actual control of the TCAM system and its resources through VTAM operator control commands. Control of TCAM's use of the resources of the TCAM system and monitoring of TCAM processes are allowed through TCAM operator control commands.

TCAM operator control consists of two system service programs supporting sets of commands which may be entered at an authorized external LU or by an application program, or from the system console, in order to examine or alter the status of the communications network resources used be TCAM during execution.

The messages that can be received following a given command are listed for each command in the *TCAM Operation* manual.

Operator control is divided into two components, called *basic operator control* and *extended operator control.* Basic operator control is activated as a subtask of the initiator and is required. TCAM's basic operator control supports a set of basic operator commands that allows you to determine the status of your TCAM system and to alter, start, or stop part or all of TCAM interface with VTAM.

Extended operator control is an optional system service program that may be started as an initiator subtask, depending on the requirements of your location. Extended operator control supports a set of extended operator commands for monitoring and controlling the TCAM resources shared by the TCAM system. With these commands, you can display information concerning external LUs, queues, and system buffer units; you can also reroute messages to an external LUs alternate destination and purge an external LUs message queue.

Both basic and extended operator commands may be entered from an application program. This section describes how to enter basic operator commands from an application program, and briefly discusses the task of entering extended operator commands.

TCAM allows you to specify an application program as a basic operator control external LU, thereby giving the program the capability to issue TCAM basic operator commands. Among other things, operator commands may be used to stop and start external LUs, interrogate the status of an external LU or queue, and load and delete IBM service aids.

Operator commands may also be entered from the system console or from an external LU that has been designated as an operator control station. However, only application-program-initiated operator commands are discussed in this manual. See the *TCAM Operation* manual for details on how to issue operator commands from either the console or an external LU. Any of the operator commands documented in the *TCAM Operation* manual may be issued through an application program. The rationale and options for each command are also documented in the *TCAM Operation* manual. They are not duplicated here.

Messages in response to operator are sent to the destination specified by the ALTDEST operand of the TPROCESS macro or to the destination specified IEDOPCTL macro. If both the ALTDEST and the RSPDEST operands are coded, the destination specified by the RSPDEST operand is used. If no command-reply destination is specified, the reply is sent to the TCAM dead-letter queue (DLQ).

The *dead-letter queue* is a destination queue assigned to accept messages with invalid addresses. (See the discussion of the DLQ operand of the INTRO macro in *TCAM Installation Reference*). If no dead-letter queue has been specified, the command-reply message is lost.

*Note:* The message that can be received in response to a given message, as well as the texts of response messages, may be changed from release to release of TCAM. You should check *TCAM Operation* and *TCAM Messages* for each release to make sure that the response messages expected in your application program have not changed.

In order for an application program to be able to issue basic operator commands, several macro operands must be specified in a TCAM message control program. Without these specifications, TCAM will not be able to distinguish a program-issued basic operator command from normal message traffic. See the "Summary of Related Operands" discussion at the end of this section.

## How to Format and Issue Basic Operator Commands

When an application program issues a basic operator command, the program must first move an image of that command into its output work area. A PUT or a WRITE macro is then issued to transfer the command from the application program to the MCP.

You are responsible for entering a control field preceding the command. Either the CODE or IEDOPCTL macros in the TCAM message handler that processes messages entered by the application program recognizes this message as an operator command. Once the command is recognized as such, it is processed like any other operator command. The command is be executed by TCAM, and an appropriate reply message is issued.

### Formatting Basic Operator Control Commands

A basic operator command has several fields. These fields must be separated by one or more blanks and must appear in the order specified in the following discussion. In addition, commands entered from an application program cannot be longer than the size of the application-program work area and, in terms of TCAM buffers, must be no longer than the buffer size as determined by the BUFSIZE operand of the PCB macro. All entries in a command issued from an application program must be issued in uppercase letters.

The format for specifying basic operator commands is:

```
  controlchars   operation   operand(specifiers)   ending
```

A correlation ID may be inserted between *controlchars* and *operation* and with intervening blanks. This ID will allow the user to correlate the request to the response. For more information, see the *TCAM Operation* manual.

*controlchars* - Must be a character string of one to eight nonblank characters conforming to the rules for assembler-language symbols. This character string must be unique in the TCAM network since it identifies this message to TCAM as a basic operator command.

The same character string must identify all basic operator commands being entered from a particular application program. It is specified either at assembly by the CONTROL operand of the INTRO macro in the MCP, or at

INTRO execution by entering a value in response to the "SPECIFY TCAM PARAMETERS" message.

In addition, user-written code in the MCP can override the CONTROL operand to change this character string at your discretion. See *TCAM Installation Guide* for details.

*operation* - One of the following operation types must be entered in this field. Separate the *controlchars* field from the *operation* field with at least one blank.

- DISPLAY - Status of a group, external LU and other activities
- HALT - TCAM system closedown
- HOLD - Messages on a particular TCAM queue
- MODIFY - Option fields
- RELEASE - Messages from a particular TCAM queue

A short form for each command is also available for your convenience. The short forms are:

| | |
|---|---|
| DISPLAY | D |
| HALT | Z |
| HOLD | H |
| MODIFY | F |
| RELEASE | A |

The examples of operator commands given in the next section show the use of both forms. Separate the *operation* field from the *controlchars* field and the *operand(specifiers)* field with at least one blank.

*operand(s)* - This field is where you must specify all the keyword operands and variable parameters that you wish to specify. These operands and their parameters detail which functional operations you wish to occur (such as DISPLAY *primary external LU name*). If more than one operand is coded, they must be separated by commas, with no intervening blanks. Brackets indicate a voluntary option; a parameter value for the enclosed operand may or may not be coded depending on your requirements. Braces indicate that one of the values in the enclosed stack must be coded. Parameters described in lowercase are called variables and must be replaced by a numeric or character value. An operand that is given in either uppercase or lowercase letters and not enclosed in brackets or braces indicates a value that *must* be coded. Uppercase values must be coded as given, lowercase operands should be replaced by a value significant to your application program.

For a description of the most common variable operands, see "The Operator and the System" in *TCAM Operation*.

Separate the *operand(specifiers)* field from the *operation* field and the *ending* field with at least one blank.

*ending* - The end-of-message or end-of-command symbol. The symbol for application programs is EOT. This is the EBCDIC EOT character (X'37'), not 'EOT'. The ending field must be separated from the operand field by one or more blanks. Any characters other than EOT appearing between a blank operand field delimiter and the first character of the ending are considered to be comments and are ignored by TCAM.

### Issuing Basic Operator Control Commands

The following two examples are typical of the circumstances under which a basic operator command might be issued from an application program and indicate the various ways an operator command may be issued. See the *TCAM Operation* manual for details on specifying commands from an external LU or the system console.

*Example 1*: If the MCP has been started and the *procname* field of the system console START command used to start the TCAM initiator specifies AQTPROC.QID and the INTRO macro in the MCP specifies CONTROL=OPID, the command to change an external LU from secondary to primary operator station status can be:

```
        OPID MODIFY QID,OPERATOR=NYC                 EOT
  or    OPID F QID,OPERATOR=NYC                      EOT
  or    OPID MODIFY AQTPROC.QID,OPERATOR=NYC         EOT
  or    OPID F AQTPROC.QID,OPERATOR=NYC              EOT
```

Where EOT= X‘37’.

*Example 2*: If the TCAM initiator is being executed as a normal job through the system input device with the jobname TCAMJOB and the INTRO macro specifies CONTROL=OPID, the commands of Example 1 may become:

```
        OPID MODIFY TCAMJOB,OPERATOR=NYC             EOT
  or    OPID F TCAMJOB,OPERATOR=NYC                  EOT
```

## Command Correlation Information Included in Extended Operator Control Replies

Reply messages from extended operator control command processors can be correlated with the command that produced the messages. This correlation is provided by information furnished with reply messages generated by extended might be useful when your application program issues several replies outstanding at the same time. Also, if an application program, in addition to issuing extended operator control commands, is designated as the extended operator control primary external LU, it may require a means to distinguish *unsolicited* notices from the replies to commands it has issued.

The correlation method used is somewhat different from what would be used with basic operator control. The following points outline how command/reply correlation is provided and what the programmer should consider.

- Extended operator control commands from extended operator control external LUs (terminals or application programs) always contain an FHP in the first (or only) command message buffer. An FHP is also included with all reply messages.

- Extended operator control command processing modules save the value of the four-byte field FHPUSERD from the FHP sent with the input command and include it in the FHP of all reply messages sent to the *originator* of that command.

- Text and formatting of commands and replies are not affected in any way by the contents of the FHPUSERD field. It is the installation's responsibility to assign, insert, extract, and use the identification value in message handlers and/or the application program in any way that suits the installation's purpose.

- Because extended operator commands can generate messages for external LUs (or programs) other than the originator, there are a few special cases to note when identification is used (when the value of FHPUSERD is not binary zeros on command input). The convention discussed below is followed so that, in addition to the reasons given above, an installation can use the FHP field of FHPUSERD for different purposes without conflict.

On commands that produce replies to another external LU (as well as the originator), FHPUSERD is preserved only on replies to the originator. The value on output to nonorigination external LUs is binary zeros.

Examples of these commands are:

- DATA ALL,destname
- QUEUE AUTO,destname, ...

On commands that produce notices to the key operator destination, the identification value on the FHPs of such notices is binary zeros.

Examples of these commands are:

- SEND
- RESEND

*Note:* Use of identification does not affect the FHPs of messages sent or resent by SEND or RESEND even if they are directed to the originator. On commands that produce notices to the key operator destination, when the key operator destination is also the originator, if identification is being used (FHPUSERD was not zero on the command), then there may be two kinds of output to the key destination:

- Replies with identification information included to the key operator destination in its role as originator
- Notices without identification information to the destination in its role as the key operator destination.

*Note:* Sometimes the text of a notice can be the same as the text of a reply. This means that duplication (aside from the value of FHPUSERD) that might otherwise be suppressed will occur when correlation information is requested.

If the installation has developed additional extended operator commands and command processors accessed via the DKJUSER table, it is the installation's choice whether to support this convention in output produced by the user's command processor.

## Initialization in the MCP for Basic Operator Control in an Application Program

To initialize the basic operator control in an application program, you must code values for several operands of the INTRO, TERMINAL, and TPROCESS macros, and CODE or IEDOPCTL macros. These macros must be coded in a message control program, not in the application program. However, an application program cannot issue operator commands unless these options have been specified in the MCP.

The CONTROL operand of the INTRO macro allows you to specify a unique set of control characters that identifies a message from your application program as a basic operator command. This same set of control characters must precede every basic operator command issued from your application program. If these control characters are omitted, the MCP will not recognize this message as a basic operator command, and the command will either go to the dead-letter queue or be lost.

Both TERMINAL and TPROCESS macros associated with the selection of an application program as a basic operator control external LU must specify SECTERM=YES to designate the program as an operator control station. Also, an application program will not receive the response messages for basic operator commands if neither the ALTDEST operand of the TPROCESS macro nor the RSPDEST operand of the CODE or the IEDOPCTL macro is coded.

## Summary of Related Operands

The following operands must be coded as specified either in an MCP or in an application program before operator commands may be issued from your application program:

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| INTRO | CONTROL | MCP | This operand specifies a symbolic label for all basic operator commands issued from your application program. The control-characters field of each basic operator command issued from your application program must contain the same character string. |
| TERMINAL | SECTERM | MCP | This operand identifies a particular external LU as a secondary control station to which TCAM will transmit replies to basic operator commands that were issued from your application program. |
| TPROCESS | SECTERM | MCP | If the TPROCESS entry associated with this macro is for PUT or WRITE macros only, this operand designates the application program as a basic operator control station that is capable of issuing operator control commands. In order to receive the ACF/ TCAM replies to basic operator commands, the ALTDEST operand for this macro must also be coded. |
| TPROCESS | ALTDEST | MCP | If you wish the TCAM reply to all the basic operator commands that you issue from your application program to be returned to the same program, you must enter the terminal-table entry label for that program here. The replies to basic operator commands are returned to whatever destination is named here. Otherwise, any response message to an operator command is sent to the dead-letter queue (DLQ); or, if no DLQ is specified, the response message is lost. |
| PUT/WRITE | | Application program | All basic operator commands issued from an application program *must be preceded* by the *controlchars* field. This field must be identical to the CONTROL=variable operand of the INTRO macro. |
| CODE | | MCP | This macro tests for basic operator control commands, translates the data, and transfers control accordingly. |
| IEDOPCTL | | MCP | This macro tests for basic operator control commands and transfers control accordingly. |

## How to Issue Extended Operator Commands from an Application Program

In order for your application program to issue extended operator commands, you must designate the program as an extended operator control station by setting a bit in the TCSOPTS option field associated with the TPROCESS entry for PUT or WRITE macros issued by the application program.

Replies to basic operator commands entered by an application program and directed to the basic operator control system service program are routed to the destination specified in the ALTDEST operand of the PUT/WRITE TPROCESS entry. Replies to extended operator commands entered by an application program and directed to an extended operator control system service program located in another TCAM are routed to the destination specified in the RETURNQ option field associated with the PUT/WRITE TPROCESS entry. In either case, the specified destination may be the name of a GET/READ TPROCESS entry associated with your application program.

The format of extended operator commands is described in *TCAM Operation* manual. Such commands are entered as messages, with no control-characters preceding them. These commands must be routed to extended operator control by forwarding them to the extended operator command TPROCESS OPRA or on an end-to-end session between the application program and extended operator control.

### Direct Forwarding to Extended Operator Control

Extended operator control requires the presence of an FHP and certain fields in the FHP initialized. If a message is to be forwarded to OPRA through DEST=PUT in the application AMH, either the application or the AMH must build the FHP. The FHP may be built by the application program and transferred as data. In which case, FHPHEADP should point to the first byte of data, and FHPMODEA should be initialized to the character ‘C’. The incoming group of AMH then issues FHPTEST ACTION=SETYES and FHPBUILD BLDOAF=YES before forwarding to OPRA. The FHP may also be built in reserved bytes by the AMH by issuing the FHPBUILD macro and then initializing FHPMODEA to the character ‘C’ before forwarding to OPRA. FHPHEADP is set by FHPBUILD.

### Forwarding Using End-to-End Sessions

If the extended operator control commands are to be routed on an end-to-end session, before entering such commands, your application program must format and transfer to the MCP a special message called a *logon message*. This logon message must have the key name defined in a KEYDEF macro and the KEYDEF must specify RESOURCE=OPCTL. The incoming group of the AMH for an application program transferring commands on an end-to-end session must create an FHP through an FHPBUILD macro and must route by key through a KEYPROC macro. The KEYPROC macro will automatically initialize the FHPMODEA field to the character ‘C’.

You may go into end-to-end session with the extended operator control system service program (SSP), in your own host or (if the extended networking capability of TCAM is being used) with the extended operator control (SSP) located in another host node.

Once you are in end-to-end session, all messages entered by an application (except for basic operator commands) are be routed to the extended operator control system service program. Basic operator commands entered on an end-to-end session with the extended operator control (SSP) in the TCAM system containing the application program will be routed to the basic operator control SSP).

To end an end-to-end session with the extended operator control (SSP), your application program formats and transfers to the MCP a special message called a *logoff message.*

The formats of the logon and logoff messages for the end-to-end session with the extended operator control (SSP) are installation dependent and should be obtained from the system programmer who designed the MCP.

If your application is going to function solely as an extended operator control station, you may avoid the necessity of entering logon and logoff messages by having the system programmer set up your TPROCESS macro so that your application program enters into a *perpetual end-to-end session* with an extended operator control (SSP) when activated. For details, see "Coding the Message Handler" in *TCAM Installation Guide.*

The KEYPROC macro provides exits that may be taken if the extended operator control (SSP) is unavailable when the logon message is sent. A message may be generated in this exit to inform your application program of this fact. Such messages may be installation-dependent.

## Inspecting and Changing TCAM Control Elements

TCAM offers three macros to the application programmer who wants to inspect or change various TCAM control elements. Two different control elements may be examined: the queue control block, and a terminal-table entry. A *queue control block* (QCB) contains control information about a specific destination queue. There is a set of queue control blocks for each queue in TCAM. For a discussion on the format and use of the queue control block, see *TCAM Program Reference Summary.* The TCAM *terminal table* contains an entry for each addressable device and application program process queue in the network. The types of entries that can be made in the terminal table are described in *TCAM Installation Guide* and *TCAM Program Reference Summary.* The three macros discussed here offer two capabilities for controlling your terminal characteristics.

1. Interrogation

   a. The TCOPY macro requests TCAM to copy the contents of a particular terminal-table entry and its associated option fields into an application-program work area.

   b. The QCOPY macro requests TCAM to copy the contents of a particular queue control block (and its related priority QCBs) into an application-program work area.

2. Modification

   a. The TCHNG macro allows you to replace a terminal table or option field entry with the contents of your application-program work area.

Protection against unauthorized use of the TCHNG macro is provided through the optional PASSWRD operand. If password protection was specified on the INTRO macro in the MCP, the same password must be specified on this macro. If a password was coded in the MCP, but not in the application program, the macro will not execute, and instead, control will go to the next sequential instruction. In addition, terminal-table entries can be protected from changes through the TCHNG macro through the use of a "field-sensitive" mask in the MCP. See the TCHNG macro discussion for details.

You are required to have at least one TCAM application-program DCB in any application program in which these macros are issued. Consultation with the TCAM system programmer is advisable before implementing any one of the macros in this group.

If the MCP with which your application program interfaces uses routing by key, you may wish to inspect and modify the fixed header prefix (FHP) associated with each message routed to your application program in this way. The following section includes information on how.

## Inspecting an Entry in the Terminal Table (the TCOPY Macro)

You may use the TCOPY macro to request TCAM to move the contents of a terminal-table entry that you have designated to a work area in your application program. The terminal-tables are for LU's. Functions similar to those provided by this macro are also provided by the Display Status and Message Number of Resource and the Display Option Field operator commands.

You must ensure that your work area is large enough to accommodate the largest possible string of data that may be moved into it by a TCOPY macro. If the work area is not large enough, main storage adjoining the work area will be overlaid.

*Note:* You may determine the length of the longest possible string of data that the TCOPY macro can move into a work area by looking at the assembler listing for the MCP. Under each TERMINAL, TLIST, TPROCESS, and LOGTYPE macro expansion are control sections having "TERMINAL ENTRY," "OPTION OFFSETS," and "DEVICE-DEPENDENT FIELDS" in their comment fields. These control sections (CSECTs) indicate the length of the terminal-table entry, the option-field offsets, and the device-characteristics fields, respectively. You should find the sum of these three lengths for each terminal-table entry you wish to copy and add to this sum the total length of the option fields associated with that entry.

The work area named in TCOPY should be at least as large as the total value obtained in this way.

The specification of this macro in an application program is optional. You should not use this macro unless it is necessary for you to examine a terminal-table entry.

A representative terminal-table entry describing a single external LU has the format depicted in Figure 4-1 on page 4-13. See the *TCAM Program Reference Summary.*

Referring to Figure 4-1 on page 4-13 in addition to the contents of the terminal-table entry itself, the TCOPY macro also moves the contents of any option fields associated with a terminal-table entry into the specified application-program work area. The first option field immediately follows the last device-characteristics field. The two-byte TRMOPTBL field, located at an offset of 18 bytes from the beginning of each terminal-table entry, contains the offset from the beginning of the entry to the beginning of the first option field. The length of the TRMOPT field (+20) is variable. If no OPTION macros are coded in the MCP, no space is allocated for the TRMOPNO field, the TRMOPTBL field, or the TRMOPT field.

A variable number of device-characteristics fields follow the TRMOPT field, or the TRMCHCIN field if no OPTION macros are coded. The first byte of each device-characteristics field contains the binary length of that field. The rest of the field contains the device-dependent data. See *TCAM Program Reference Summary* for a discussion of the individual terminal-table field names and their contents.

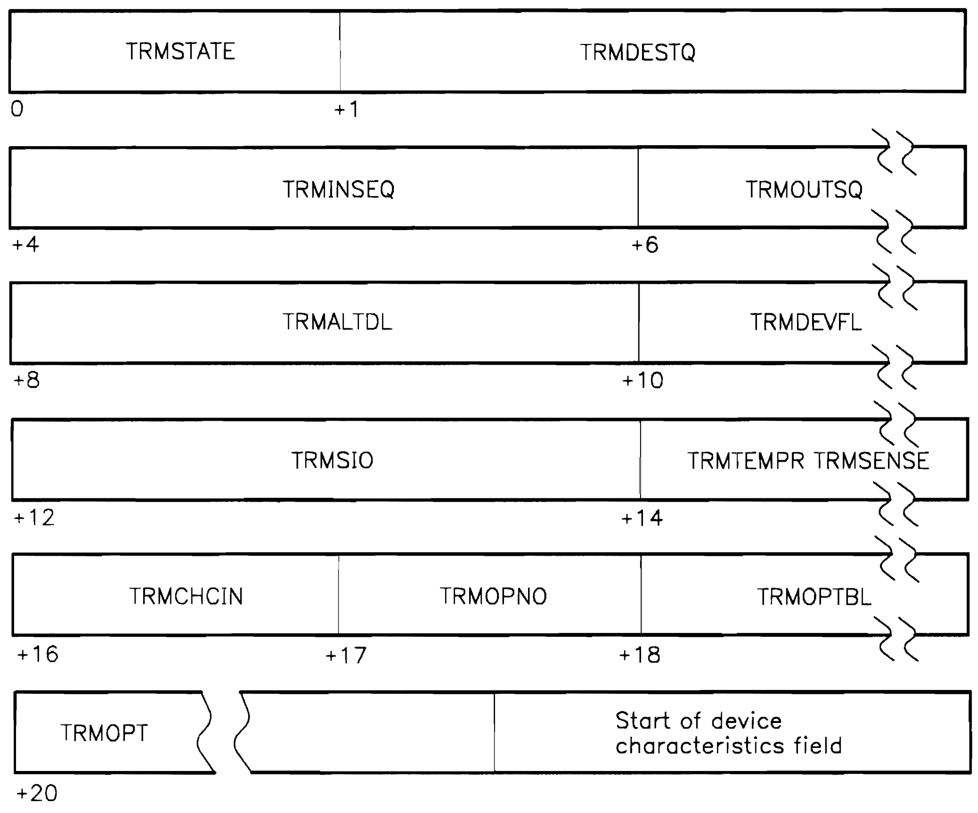


**Figure 4-1. The Length of the Terminal Table Entry DSECT**

## Inspecting a Destination Queue Control Block In the MCP (the QCOPY Macro)

You may use the QCOPY macro to request TCAM to copy the contents of a destination or master queue control block (QCB) and its related priority QCBs (if any) into a designated application-program work area. This copy function may either be coded to execute only after a specific number of messages have been queued, or it may be coded to execute unconditionally. You can specify a threshold number of messages on the QCOPY macro for the case where you want the QCOPY macro to execute conditionally. The QCOPY macro may be issued for a host LU or an outboard LU. Only the master QCB will be copied if there is no priority QCB.

The QCB is a TCAM control block associated with a particular destination message queue. For a complete description of queue control blocks, see the "Data Areas" section in *TCAM Program Reference Summary.* A master QCB is 40 bytes long and always has at least one priority QCB associated with it, even if no priorities are specified.

Each priority QCB is 28 bytes; therefore, the formula for determining the number of bytes needed in the application-program work area for a given QCB is:

68 + 28n bytes

Where *n* is the number of different priorities specified for the external LU whose associated QCB is being copied. The number of allowable priority levels is specified by the LEVEL operand of the TERMINAL macro for the related external LU.

Part of the function of QCOPY is also provided by the Display Queue Control Block operator command. This macro is optional in an application program and should not be coded unless it is necessary for you to examine a queue control block.

## Changing an Entry in the Terminal Table (the TCHNG Macro)

You may use the TCHNG macro to request TCAM to replace the contents of a particular terminal-table entry with the contents of your application-program work area. See the preceding discussion on the TCOPY macro for a description of a typical terminal-table entry. Table entries for a host or an outboard LU may be modified based on the mask defined on the INTRO macro. Option fields associated with a terminal-table entry may also be modified by this macro.

All the fields necessary for proper execution of TCAM must be placed in the terminal-table entry in proper form. If you use this macro, you are responsible for maintaining the integrity of the terminal-table entry that you change. The storage for each terminal-table entry's device dependent field and option field is allocated at assembly time. If a new field is added to either the device dependent field or the option field, storage following the field is destroyed. This also occurs if the length field of a device dependent field is increased. You should, therefore, not add any new device dependent or option fields, or increase the length of any field.

TCAM provides a field-sensitive mask to allow the system programmer to protect terminal-table entries from inadvertent modification or destruction by an application program. These masks allow the application program to alter only certain fields in a terminal-table entry. The masks will apply to device classes, not specific devices. Details on how a field-sensitive mask is specified are given in the description of the OPMASK operand of the INTRO macro discussion in the *TCAM Installation Reference.*

Masks may be established at TCAM generation to restrict the kinds of fields that may be modified by a TCHNG macro. Masks may be specified for log entries, LUs, list entries and TPROCESS entries.

The contents of option fields may also be modified by the Insert Option Field Data operator command. The TCHNG macro is optional in an application program. You should not use this macro unless it is necessary for you to make changes to a terminal-table entry.

### Summary of Related Operands for the Copy and Change Macros

Values for the following macros and macro operands must be coded in the MCP as specified before the referenced macro will execute properly in an application program.

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| INTRO | PASSWRD | MCP | If this operand is coded in the MCP, the TCHNG macro will not execute unless it also specifies the same password. |
| INTRO | OPMASK | MCP | Used to specify a set of security masks to protect particular fields in a terminal-table entry. This facility is used in conjunction with the TCHNG macro. |
| TERMINAL | LEVEL | MCP | The length of a QCB to be copied can be calculated by the formula 68 + 28n bytes, where *n* is the number of priorities specified by this operand. |

## Inspecting and Modifying the Contents of a Fixed Header Prefix

If the system programmer elects to pass the full FHP to the application program, it is placed in the program's input work area along with the message as follows:

```
FHP      DATA           AREA
```

TCAM provides a macro instruction (DKJFHD) which generates a dummy section (DSECT) describing the FHP. Using the DSECT and assuming the name of the program's work area is AREA, use the following instructions to access the address of the first data character of the message:

```
LA       6,AREA         Establish addressability
USING    FHPSTART,6
SR       1,1            Clear a register
IC       1,FHPHEADP     Get offset to data
LA       2,0(1,6)       Put address of data in register 2.
```

The FHP field FHPHEADP contains a value which, when added to the address of the first character of the FHP, yields the address of the first data character. FHPHEADP is an offset, then, to the data from the start of the FHP.

FHPOAFLD is the symbolic name of the field in the FHP which contains the network address of the message origin (the origin address field or OAF).

If any field in the passed FHP is to be referred to, it should be referred to by its symbolic field name.

The DKJFHD macro instruction must be issued in the application program to obtain the necessary dummy section. The macro is described in Chapter 5.

The contents of the FHP are discussed in "Designing the Message Handler" in *TCAM Installation Guide*. The DSECT for the FHP is described in the *TCAM Program Reference Summary*.

## Coordinating TCAM Checkpoints of the MCP with Checkpoints of an Application Program

When you are updating data files by sending messages from an application program, coordination of the output messages with messages being sent to the application program following a restart can be achieved by using a combination of MVS checkpoints and the TCAM CKREQ macro. This facility would be used in conjunction with "flip-flop" data sets that are set up to change back to their status as of the last MVS checkpoint upon restart.

TCAM checkpoints of the message control program can be coordinated with MVS checkpoints of TCAM application programs by issuing a CKREQ macro in the application program. The purpose of this coordination is to allow the MCP and each application program to restart at the same point following system failure. This section describes how the CKREQ macro can ensure coordination between the application program and the MCP, and how a user-specified exit from an input or output DCB macro in an application program may be used for this purpose. The MVS advanced checkpoint/restart facility for application programs is described in *Checkpoint/Restart.*

Another way to obtain coordination upon restart is to specify CKPTSYN=NO in the TPROCESS macros for the application program, and take an MVS checkpoint each time a data file update occurred. If one data file update per message were performed and one MVS checkpoint per message were taken, upon restart the application program would only have to check for one duplicate message in order to ensure that updating of the data file resumes from the point of interruption.

In this discussion on restarts, system failure is assumed to involve MCP failure. If the MCP fails, the application program is abnormally terminated by TCAM and, consequentially, all of its data sets are automatically closed. Therefore, after the MCP is restarted, you must also restart your application program and reopen all necessary data sets.

Failure of an application program need not be accompanied by failure of the MCP. However, in some application programs, you might wish to close down the MCP following abnormal termination of an application program, so that both might be restarted from the same point. See the following discussion on coordinating MCP and application program restarts for more on this topic.

## The CKREQ Macro

When a CKREQ macro is executed in an application program, a checkpoint request is written in the TCAM checkpoint data set for each queue to which a GET or READ macro can be directed by that program. If restart is necessary, this checkpoint request record is used to update the MCP environment. The checkpoint/restart function causes transmission to the application program after restart to begin with the last message marked serviced at the time the last checkpoint request record was written, rather than with the last message marked serviced before the MCP closedown or failure.

When a restart is performed after the CKREQ macro has executed, normal processing of the TCAM message queues does not occur if CKPTSYN=YES was specified on the associated TPROCESS macro. Instead, the first message to be sent from the MCP to the application program following restart is determined by the contents of the last checkpoint request record written for that queue as the result of execution of a CKREQ macro. If CKPTSYN=NO was specified, the first unserviced message in the highest-priority group of messages on the message queue for that application program is sent following a restart.

When the CKREQ macro is used in an application program with low message traffic, the checkpoint request record written as a result of execution of the macro may be obsolete when compared to the MCP environment. For example, it may contain information pertaining to a zone on a reusable disk that has already been reformatted. When this happens, messages can be lost. Therefore, consideration should be given to the matter of how often checkpoints should be taken, and how application program and MCP checkpoints should be coordinated.

After processing a reasonable number of messages or records (see the checkpoint discussion in *TCAM Installation Guide)*, the application program should take an MVS checkpoint. An MVS checkpoint cannot be taken from an application program that is attached to the TCAM MCP as a subtask. Immediately after the MVS checkpoint is taken, the program should issue a CKREQ macro. If this sequence is followed, upon restart after an MCP failure, the application-program environment will be restructured using the latest MVS checkpoint, and no more than the number of messages processed by the application program since the previous checkpoint will be sent.

*Note:* If both MVS checkpoints and TCAM checkpoint request records are used, a CKREQ macro should be issued each time an MVS checkpoint is taken.

Figure 4-2 on page 4-18 shows how to use the CKREQ macro for checkpoint coordination.

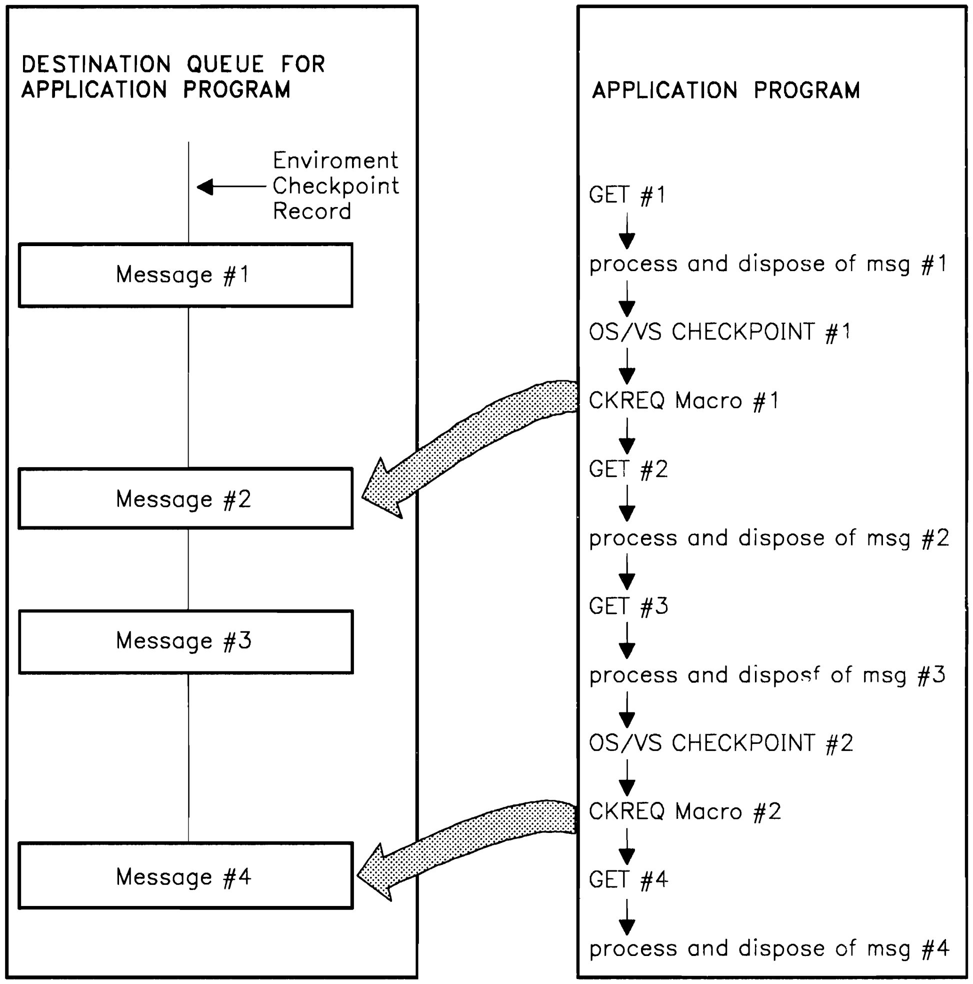

DESTINATION QUEUE FOR APPLICATION PROGRAM
Enviroment Checkpoint Record
Message #1
Message #2
Message #3
Message #4
APPLICATION PROGRAM
GET #1
process and dispose of msg #1
OS/VS CHECKPOINT #1
CKREQ Macro #1
GET #2
process and dispose of msg #2
GET #3
process and disposf of msg #3
OS/VS CHECKPOINT #2
CKREQ Macro #2
GET #4
process and dispose of msg #4

**Figure 4-2. Using the CKREQ Macro for Checkpoint Coordination**

Referring to Figure 4-2, a TCAM environment checkpoint record is written before the first GET macro is issued. After the first message is processed and disposed of by the application program, an MVS checkpoint is taken. Upon return from the checkpoint subroutine, a checkpoint request record is written to record the status of the destination queue for the application program by using the CKREQ macro. When the next GET macro is satisfied (that is, after the second message has been moved into the work area), the first message is marked serviced in the destination queue. When the third GET macro is satisfied, the second message is marked serviced. After the third message is processed by the application program, another MVS checkpoint record and TCAM checkpoint request record are written. When the fourth GET macro is satisfied, the third message is marked serviced.

Assume that a system failure (failure of the MCP) occurs during the processing of the fourth message. In this case, upon restart, the application-program environment is reconstructed using the second MVS checkpoint and the fourth message (the message pointed to by the second CKREQ macro) is the first message sent upon restart. No duplicate messages would be sent to the application program from the destination queue.

Now, assume that system failure occurs during processing of the third message. In this case, the application-program environment is reconstructed using the first MVS checkpoint, and message number 2 is the first message sent upon restart. This is the next unprocessed message with respect to the reconstructed application-program environment.

As a final example, assume that system failure occurs after the second MVS checkpoint is taken, but before the second CKREQ macro is executed. In this case, the application-program environment is reconstructed using the second MVS checkpoint, but the first message sent upon restart is message number two. Messages number two and three are duplicate messages with respect to the reconstructed application program environment.

## Using the DCB Exit for Checkpoint Coordination

The input and output DCB macros in TCAM application programs permit specification of a user-written routine to take an MVS checkpoint after each TCAM environment is taken. You may designate the address of an exit routine that you must code by specifying a value for the EXLST operand on the input or output DCB macro for your application program. (See the DCB macro discussions in Chapter 5). The format and contents of the exit routine list are discussed in *Data Management Services*.

You should specify your own checkpoint routine in the exit list by coding an X'OF' as a control byte in the exit list and by following the control byte with the three-byte address of a checkpoint routine that you have coded. The routine must save and restore the contents of registers 1 and 14. You must not store data in the area pointed to by register 13 upon entry to this routine. All registers except 1, 13, and 14 will contain what they did before the macro causing the exit to be taken is executed. In addition to coding the EXLST operand of the input or output DCB macro, you should also specify CKPTSYN=YES in the TPROCESS macro for each process queue you wish to checkpoint.

When the EXLST operand is coded in an application program DCB, an indication is made to the application program each time an environment record is made. If your checkpoint routine is coded in the input DCB macro, the first GET or READ macro issued by the application program after the TCAM environment checkpoint is taken passes control to your checkpoint routine. Your routine is then responsible for taking an MVS checkpoint before returning to the calling routine. The GET or READ macro is not satisfied until after the calling routine regains control.

If the checkpoint routine is coded on the EXLST operand of an output DCB macro, the first PUT or WRITE macro issued by the application program after the environment checkpoint is taken passes control to your checkpoint routine. Again, the PUT or WRITE macro is not satisfied until after control is returned to the application program. If MVS checkpointing is used, a CKREQ macro should be issued after each operating system checkpoint.

Upon restart following a system failure, message traffic to the application program resumes with the message in each queue that was the earliest completed, unserviced message in the highest priority group at the time the last checkpoint was taken. All other unserviced messages on the queue at the time the environment checkpoint was taken are ultimately sent to the application program upon restart.

By coding a value for the CPINTVL operand of the INTRO macro in the MCP, the system programmer may ensure that the TCAM environment checkpoints are taken within a specified time limit after the MVS checkpoint.

*Note:* Ordinarily, the MVS checkpoint routine cannot be invoked from a DCB exit routine. When the DCB involved is a TCAM input or output DCB, however, this restriction does not hold.

## Coordinating MCP and Application-Program Restarts

When restarting an MCP in conjunction with an application program, the MCP must be restarted first. Then the application program may be restarted using MVS restart facilities.

If the MCP terminates abnormally, TCAM application programs currently active are also automatically halted abnormally, providing there was at least one open TCAM DCB. If the TCAM checkpoint facility is being used, a point-of-last environment restart may be performed for the MCP; as before, the application program may then be restarted.

If the application program ends abnormally and the MCP does not end, you have two courses of action open to you:

1. You may restart the application program without closing down or ending the MCP job. In this case, the first message received by the restarted application program from a particular process queue is the unserviced message in the highest-priority group for that queue that was completely received and queued before any other message in the highest priority group of that queue.

   *Note:* A message is not marked serviced until the next message to be sent to the application program from the same queue has been transferred in its entirety to the application program.

   Message A is not marked serviced until message B has been entirely transferred to the application-program work area. If the closing of an application program data set is invoked either by the CLOSE macro or by an abnormal termination, message B is marked serviced if it has been completely transferred to the application program prior to close. Therefore, if message A is transferred to the application program and is followed immediately by message B on the same process queue, and if the application program terminates abnormally when half of message B has been transferred to the application program, the first message to be transferred to the application program following its restart would be message B. If this course of action is followed, no synchronization of MVS checkpoints with the TCAM MCP checkpoints is performed. If the final segment of message B is being transferred to the application program and a failure occurs during processing, message B is marked serviced by the close routine activated by abnormal termination processing. Message B is not re-sent when the application program is restarted. If necessary, the message may be reprocessed by using the POINT macro and the TCAM message-retrieval facility.

2. Following failure of the application program, you can close down or terminate the TCAM MCP, and then reactivate the MCP with a point-of-failure or point-of-last-environment restart, and the application program by an MVS restart. In this case, the application program will receive from each process queue those messages that were on the queue and unserviced at the time that the last checkpoint request record (or environment record, if no checkpoint request record was made) was written for that queue, plus all messages that were placed on the queue after the last checkpoint request record was written.

When reusable disk queuing is used, there is an advantage to be gained by combining the two coordination methods described here by issuing both a CKREQ macro and an MVS checkpoint request in the DCB exit routine. If an environment checkpoint is taken due to a zone changeover on the reusable-disk data set, checkpoint request records written before the data set reorganization are now out of date, because they do not point to the disk zone currently being used. Since the DCB exit routine is given control after each environment checkpoint is taken, it provides the user with an opportunity to write a fresh checkpoint request record after each zone changeover.

### Checkpointing Operator Control Commands

If you intend to issue operator commands from an application program, you should first be familiar with the *TCAM Operations* manual. If the checkpoint DCB in the TCAM MCP has been opened, incident records are written when certain operator commands are successfully processed. A list of the MODIFY commands that cause incident records to be written can be found in the *TCAM Operation* manual.

None of the display commands and no unsuccessful, operator-initiated commands are checkpointed.

### Summary of Related Operands

The following macros or macro operands must be coded either in an MCP or in an application program before the TCAM checkpoint/restart facility can be implemented.

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| TPROCESS | CKPTSYN=YES | MCP | The CKREQ macro cannot be issued in an application program unless this operand is coded in the MCP. |

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| INTRO | STARTUP | MCP | Specifies the restart options you wish for any TCAM restarts. |

The following macros and operands may be specified optionally, depending on your circumstances.

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| TPROCESS | CKPTSYN=NO | MCP | If you want to take your own MVS checkpoint in the application program each time a file update occurs. |
| DCB | EXLST | application program | If you desire to take your own MVS checkpoint each time a file update occurs. EXLST must specify a routine that you have written and included in the application-program region. |
| INTRO | CPINTVL | MCP | You may specify a time interval within which an environment checkpoint must be taken following an MVS checkpoint. |

## Determining How Many Messages are on a Specific Queue (the MCOUNT Macro)

At times, it may be necessary for you to find out how many completed messages are waiting to be processed by a specific application program. Execution of the MCOUNT macro can give you this information. MCOUNT should be issued in an application program just before a GET or a READ macro to determine how many messages are queued for a particular DCB.

The message count returned in register 1 as a result of execution of an MCOUNT macro is for the queue associated with the DCB operand of the MCOUNT macro. MCOUNT returns a message count only for an input queue of an application program. Refer to the QCOPY macro for message counts on other queues. If the DCB operand specifies an output data control block, register 1 will contain a zero after the MCOUNT macro executes.

You must issue an OPEN macro for the DCB that you intend to refer to before issuing an MCOUNT macro. The MCOUNT macro uses standard register linkage. (See the coding details for the MCOUNT macro in Chapter 5.)

The execution of this macro is not dependent on any macros in the MCP.

## Identifying Application-Program Input Messages by Time and Date Received (the TPDATE Macro)

Execution of the TPDATE macro causes the transfer of time-stamped and dated messages into your application program. However, before you can execute the TPDATE macro, the system programmer must have specified in the MCP that you wish all the messages coming into your program to be stamped and dated. (See the following "Summary of Related Operands" for TPDATE.)

The TPDATE macro allows you to:

- Specify which TCAM input message queue you wish to access
- Define a 16-byte work area in your application program where you want the time and date information to be returned
- Specify whether you wish TCAM to return an image of the current record delimiter to your application program
- Specify whether you wish record delimiters to be deleted from each message before the message is transmitted to your application program.

You must issue an OPEN macro for an application-program DCB before issuing a TPDATE macro referring to that DCB. The TPDATE macro uses standard register linkage.

### Summary of Related Operands

The following macros and macro operands must be coded in the MCP before the TPDATE macro executes your program.

| Macro | Operand | Where Macro is Issued | Notes |
|---|---|---|---|
| PCB | DATE=YES | MCP | TPDATE will not execute unless this operand is coded. |
| TPROCESS | DATE=YES | MCP | TPDATE will not execute unless this operand is coded. |

## Releasing Messages from TCAM Queues (the MRELEASE Macro)

You may use the MRELEASE macro in your application program to release all queued messages that have been intercepted for a specific external LU, LU, or application program. An intercepted LU, external LU or application program is one that has had all message traffic to it stopped. This may have occurred through execution of the Intercept a Station operator command or the HOLD macro. This macro has the same effect as the Release Intercepted Station operator command. (See the MRELEASE macro discussion in Chapter 5 for coding details.)

**Summary of Related Operands**

If the PASSWRD operand was coded on the INTRO macro of the MCP, the MRELEASE macro must specify that same password.

## Displaying Main Storage from Within an Application Program (the COREDSP Macro)

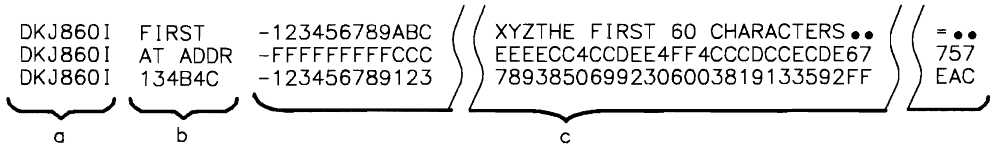
The COREDSP macro permits the display, at the system console, of selected main storage locations in the application program region. Optionally it allows the console operator to indicate whether the application program should attempt to process the current message or should discard it and obtain another message for processing.

A main storage display consists of a message formatted as three lines that are sent sequentially to the system console (see the following example). Each of the three lines contains three data fields:

1. The first field is the standard MVS message identification and type field. This field contains identical data for all lines of all display messages, namely, DKJ860I.

2. The second field contains different data in each of the three lines of a display message. In the first line, this field contains the identifier coded in the NAME operand of the COREDSP macro, or, if no NAME operand was specified, the name of the control section in which the COREDSP macro was issued. In the second display line, this field contains a constant caption, AT ADDR. In the third display line, this field contains the starting address of the main storage display in hexadecimal format.

3. The third field in each line is composed of a hyphen followed by 60 positions of display data from the area in main storage. In the first line, the data area displays the storage area as EBCDIC characters with a period (.), substituted for each unprintable byte configuration. In the second line, the data area displays the hexadecimal zone portion (the first four bits) of each byte in the storage area. The last line of a display message contains the hexadecimal numeric portion (the last four bits) of each byte of the sixty-byte main storage area being displayed.



```
DKJ860I  FIRST    -123456789ABC  XYZTHE FIRST 60 CHARACTERS••  =••
DKJ860I  AT ADDR  -FFFFFFFFFCCC  EEEECC4CCDEE4FF4CCCDCCECDE67  757
DKJ860I  134B4C   -123456789123  789385069923060038191335 92FF  EAC
```

Example of a display of a main storage message (abbreviated)

a. standard MVS message identification and type code field
b. display area identification data field
c. display data field. (In an actual display, 60 bytes are displayed.)

If the COREDSP macro is not coded with CONV=YES specified, the function of the macro is completed after the main storage display. Register 15 contains a return code of 0. The only exception to this procedure occurs when the macro specifies an invalid main storage address; in this case, the error message—mentioned below—is issued and, subsequently, the console operator is able to correct it.

If the COREDSP macro is coded with CONV=YES, an operator conversation message is sent to the system console after the initial main storage display. This conversation message consists of an MVS message in the following format:

```
DKJ871D nameid    -STARTUP-ENTER GO,NO,OR AD=
```

Where *nameid* is the name specified by the NAME operand coded in the COREDSP macro, or if no NAME operand is specified, the name of the control section in which the COREDSP macro was issued.

The operator's reply determines further processing. If the operator replies GO, he is indicating that the application program should continue processing the last record accessed. This indication is passed to the application program in the form of a 0 return code in register 15. An operator reply of NO indicates that the application program should not attempt to process the last record accessed, but should, instead, access another record to avoid repetition of the abnormal termination of the application program that resulted in the application program restart. The indication is passed to the application program as a 4 return code in register 15. The application program must interrogate the return code from the COREDSP macro and take appropriate action.

If the system console operator wishes to examine other areas of main storage before making his GO/NO decision, he may reply AD followed by either MORE or a main storage address. AD=MORE indicates that he wishes to see a main storage display of the next sixty positions after the sixty-position area previously displayed. AD=nnnnnn, where *nnnnnn* is a hexadecimal main storage address of up to six digits, generates a main storage display, in the format described previously, of the sixty-byte area beginning at the specified address. The operator may request as many displays of main storage as he requires until he has enough information to make a GO/NO decision; the conversation message, described above, is printed on the console after each main storage display.

If the main storage address specified in either a COREDSP macro or in an operator AD=reply is not a valid main storage address, or if a sixty-character display would extend to an invalid address, an error message is sent to the system console formatted as follows:

```
DKJ862I nameid - nnnnnn IS AN INVALID CORE ADDRESS
```

Where *nameid* is the name specified by the NAME operand coded in the COREDSP macro; or if no NAME is specified, the name of the control section in which the COREDSP macro was issued, and *nnnnnn* is the six-digit hexadecimal main storage address prompting the error message. This invalid address message is followed by a transmission of the operator conversation message described above; this allows the operator to request a display starting at another main storage address, or to terminate the main storage display processing via the GO/NO reply options.

## Converting Numbers into Binary, Decimal, and Hexadecimal Formats (the TCBINCNV Macro)

The TCBINCNV macro allows you to convert a halfword binary number to the equivalent principal decimal or hexadecimal value, and to convert an EBCDIC decimal number to the equivalent halfword binary number. For details, see the description of this macro in Chapter 5.

# Chapter 5. TCAM Application Programmer's Macro Reference Guide

This chapter is a reference guide for coding your TCAM application programs. The macros are arranged in this chapter in alphabetical order for ease of reference. Coding details for the following macros are included in this chapter.

| | |
|---|---|
| CHECK | OPEN |
| CKREQ | POINT |
| CLOSE | PUT |
| COREDSP | QCOPY |
| DCB(input) | QRESET |
| DCB(output) | READ |
| DKJFND | TCBINCNV |
| GET | TCHNG |
| MCOUNT | TCOPY |
| MCPCLOSE | TPDATE |
| MRELEASE | WRITE |

Details for any TCAM macro not listed here can be found in the *TCAM Installation Reference.*

The *TCAM Installation Reference* contains the macro coding details for all of the TCAM system macros used in the MCP. This chapter describes only those macros that may be used in TCAM application programs.

In order for an application program to execute in the TCAM environment, it must contain a minimum of an OPEN macro, at least one DCB macro, a GET, READ, PUT, or WRITE macro, and a CLOSE macro. Any of the other macros in this chapter are optional, and their use will depend on the application being coded.

The rationale for coding any of these macros is not discussed in this chapter. The following pages consist of "how to code" detail only. By the time you are ready to use this chapter, you should understand fully why you want to use a particular macro. However, if you still have some doubt about the proper instruction to use, you may review the topic in question in the first four chapters of this manual or discuss your problem with the TCAM system programmer.

Register notation may be used in several of these macros. When an operand is specified as a register, the application program must have inserted the value or address to be used into the register as follows:

- If the register is to contain a value, it must be placed in the low-order portion of the register unless the macro instruction description states otherwise. Any unused bits in the register should be set to zero.
- If the register is to contain an address, the address must be placed in the low-order three bytes of the register, and the high-order byte of the register should be set to zero.

## CHECK Macro

The CHECK macro:

- Checks the DECB associated with a READ or WRITE for successful completion
- May cause an application program to be placed in the wait state
- Should be issued after each READ or WRITE macro in the same order as the READ or WRITE macro instructions were issued

Refer to Chapter 3 for details.

| **Name** | **Operation** | **Operands** |
|---|---|---|
| `[symbol]` | `CHECK` | `decbname` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within a program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`decbname`

*Function:* Specifies the symbolic name of the data event control block referred to by the READ or WRITE macro instruction that this macro is intended to CHECK.

*Format:* Must be the same symbolic name as that specified in the *decbname* operand of the associated READ or WRITE macro. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number must be enclosed in parentheses and the address of the DECB must have been previously loaded into that register. Any of the general registers 1 through 12 may be used.

## Return Codes

The following return codes may be passed back to your program in register 15 after this macro executes:

| Bit | Meaning |
|---|---|
| **X'00000000'** | The CHECK macro executed successfully. |
| **X'00000004'** | End of file has been reached, and no EODAD was specified. |
| **X'00000008'** | An error (work area overflow or sequence error) occurred during execution of the associated READ or WRITE macro, and no SYNAD routine was specified. |
| **X'0000000C'** | A message was not found during retrieve (READ with POINT) or an invalid destination was specified (WRITE). |
| **X'00000010'** | The WRITE macro just issued cannot execute because of a congested destination message queue data set in the MCP. |

Completion codes that may be returned to the event control block of the DECB associated with this CHECK macro, or with a second READ or WRITE macro are:

| Code | Meaning | Comments and TCAM Action |
|---|---|---|
| X'01000000' | The read-ahead queue is empty, but the destination queue is not empty. | See comments for X'02000000'. |
| X'02000000' | An end-of-queue condition has been encountered. No SETEOF macro was specified in the MCP, and no data is currently on a read-ahead or destination queue for this DCB. | Along with X'01000000', this code indicates that the process queue has no data on it. TCAM waits for a message to be put on the read-ahead queue and then changes the value from X'01000000' or X'02000000' to X'40000000'.<br><br>*Note:* Neither the wait nor the complete bit in the ECB field of a DECB is set to 1 by the two *empty-queue* completion codes (X'01000000' and X'02000000'). This is so you can continue processing without first having to set the wait bit in the ECB to zero. You can code to test the ECB before issuing a CHECK macro. If the ECB contains a X'01000000' or X'02000000', your routine may engage in some other program activity rather than immediately issuing the CHECK macro and entering a wait state. |

| Code | Meaning | Comments and TCAM Action |
|---|---|---|
| X'40000000' | There is now data on the read-ahead queue. | A READ macro was issued previously and the input queue was empty; since then, some data has been placed on the queue. Another READ or a CHECK macro should now be issued to bring this data into your work area. TCAM reads a message into your work area and returns control to the application program at the instruction following the READ or CHECK macro. |
| X'50000000' | A READ macro was issued in conjunction with a POINT macro in order to retrieve a message; no message was found on the specified message queue. If a sequence number greater than the highest known to TCAM is used (for example, greater than 9999 or greater than the highest for the queue), this return code is not issued and the subsequent READ will get X'50' in the ECB (that is, ECBNOMSG, Message not found). If multiple retrieval was requested, this code means that the function has been terminated and the chain of messages eligible for retrieval has ended. | TCAM issues a return code of X'0000000C' in register 15. If multiple retrieval was requested, a POINT macro with a blank address field to terminate is not required and must not be issued now because the function has already terminated. |
| X'52000000' | There was a work-area overflow in the application program at the completion of the latest READ macro; or an invalid BLKSIZE operand was detected for a WRITE macro. | The CHECK macro gives control to the SYNAD exit routine in both cases. This code is also returned if the BLKSIZE operand the DCB macro is invalid (the BLKSIZE operand is greater than the APWAS operand of the INTRO macro. LRECL=0 and RECFM=F have been specified on the DCB associated with this READ macro. TCAM either (1) takes the user SYNAD exit or (2) returns a code of X'00000008' in register 15 if no SYNAD exit was specified. |
| X'54000000' | An invalid message destination was specified on a WRITE macro. | The CHECK macro gives control to the SYNAD exit because the message destination specified was not found in the terminal name table. TCAM either (1) takes the user's SYNAD exit, or (2) returns a code of X'0000000C' in register 15 if no SYNAD exit was specified. |

| Code | Meaning | Comments and TCAM Action |
|---|---|---|
| X'58000000' | A work-unit sequence error occurred. | The CHECK macro gives control to the SYNAD exit. The output DCB specifies OPTCD=C and the work-unit position field specifies the wrong type of work unit. For example, the last work unit processed was the last segment of a message, and the position field did not reflect that it was the last portion of a message. TCAM either (1) takes the user's SYNAD exit, or (2) returns a code of X'00000008' in register 15 if no SYNAD exit was specified. |
| X'5C000000' | There is a congested destination message queue data set. | The TCAM message queue data set is receiving more traffic than it can currently handle. Therefore, TCAM cannot accept your work unit at this time. This is in keeping with the TCAM approach to slowing down input because of a temporary threshold condition. You should issue the WRITE again. TCAM returns a code of X'00000010' in register 15. |
|  | The WRITE macro that you just issued cannot execute because simultaneous WRITES were issued against the same PCB. | You should issue the WRITE again. TCAM returns a code of X'00000010' in register 15. |
| X'5E000000' | A TCAM quick close-down was begun. The WRITE macro that you just issued has been rejected. | TCAM either (1) takes the user SYNAD exit, or (2) returns a code of X'00000004' in register 15 if no SYNAD exit was specified. |
| X'70000000' | A SETEOF macro was just executed in the MCP. The program work area will not contain a work unit until another read is issued. The EODAD operand and the STOP operand of the DCB macro are not coded and an MCP closedown is in progress. | The CHECK macro gives control to the EODAD exit. TCAM returns a code of X'00000004' in register 15 if no EODAD exit was specified. |
| X'7F000000' | Normal completion of a READ or WRITE macro. | Application program execution continues normally at the instruction following the CHECK macro. |

## CKREQ Macro

The CKREQ macro:

- Causes a checkpoint request record to be written in the TCAM checkpoint data set for each queue to which a GET or READ macro can be directed by an application program

Refer to Chapter 4 for details.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | CKREQ | (no operands) |

symbol

*Function:* Allows symbolic addressing of this instruction within a program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

*Note:* The CKREQ macro has no operands. This macro does not move messages. It only identifies messages that have already been transmitted. The checkpoint request record written at the execution of CKREQ macro is used to update the MCP environment if restart is necessary. The first message sent after restart of MCP is the last message marked serviced by the CKREQ macro. Registers 0, 1, 14, or 15 may be altered during execution of the CKREQ macro.

### Return Codes

One of the following return codes will be passed back to your program in register 15 after the CKREQ macro executes.

| Code | Meaning |
|---|---|
| **X'00000000'** | A checkpoint-request record has been successfully written on disk. |
| **X'00000004'** | No checkpoint-request record was written on disk for this request. |

## CLOSE Macro

The CLOSE macro:

- Is issued in the application program to deactivate any open input or output data sets.

Refer to Chapter 2 for details.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | CLOSE | (dcbname,,...) [,MF={L}] |

symbol

*Function:* Allows symbolic addressing of this instruction within a program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

dcbname

*Function:* Specifies the symbolic name of the data control block for the logical data set you wish to close with this instruction. The ellipsis in the format indicates that you may specify multiple *dcbname* parameters with this operand.

*Format:* Register notation may be used. Each *dcbname* must conform to the rules for assembler language symbols, and must be the same as the label of the DCB macro associated with the data set being closed. If more than one data set is being closed, the *dcbnames* must be separated by double commas. For example, if you were using DCB names, you would code (DCBA,,DCBB,,DCBC). Using register notation, you would code ((4),,(5),,(6)) if the addresses were in registers 4, 5, and 6.

*Default:* None. Specification of this operand is required except when a list name is specified in the execute form macro.

*Note:* Every data set in an application-program can be closed with one CLOSE macro by including the name of each of their data control blocks as an operand. If register notation is used, the address of the appropriate data control block must have been previously loaded into the register specified.

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

*Function:* Specifies whether this instruction is to be an executable instruction (MF=E), or a list instruction (MF=L) which must be referred to by an executable CLOSE macro instruction. See the *Data Management Macro Instructions* manual for the definition and use of the list and execute forms of this macro.

*Format:* L or (E,*listname*) where *listname* is any arbitrarily assigned character sequence of up to eight characters that is unique within the program. *listname* refers to the label of a list form of the CLOSE macro that contains a parameter list and resides in the constant area of your program.

*Default:* MF=E

*Note:* MF=L creates a parameter list; no executable code is generated. You must specify this parameter list among your program constants. The *dcbnames* in the parameter list are not used until the application program issues a CLOSE macro with an MF=(E,*listname*) operand that refers to a list form of the macro. The label specified by *listname* is the label of the

MF=L version of the CLOSE macro that you want executed. The MF=(E,*listname)* form of this macro executes the TCAM close routine, using the parameter list referred to by *listname*. This list was created by a macro having the MF=Loperand specified. Parameters specified in a macro having the MF=(E, *listname*) operand override corresponding parameters in the list form.

### Return Codes

None.

## COREDSP Macro

The COREDSP macro:

- Provides assistance for application program restart through displaying memory content of a specific location.
- Is normally issued in an application program, but may be issued (for main storage display during testing) anywhere in an executable section of the MCP.

The COREDSP macro instruction allows the system console operator to display selected main storage locations, and, if so specified, to indicate whether an application program should attempt to process the current message or should access another message. The issuing program may display any location in its MVS region.

You can code the macro instruction in such a way that, after an initial main storage display, the system console operator may request another display or decide that the record last accessed by the application program should not be processed again.

The contents of registers 1, 14, and 15 will be destroyed by the COREDSP macro instruction. Register 13 is presumed to contain the address of an 18-fullword register save area. All other registers are saved and restored by the COREDSP macro. When control is returned to the user: register 1 contains the address of the work area used by the macro (see the WORK operand, below) and register 15 contains the return code (0 if the operator entered GO or if no operator conversation occurred, and 4 if the operator entered NO).

Refer to Chapter 4 for details.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | COREDSP | ADDR={name }<br>{(register)}<br>[,CONV={YES}]<br>{NO }<br>[,NAME={name }]<br>{(register)}<br>[,WORK={name }]<br>{(register)} |

```
symbol
```

*Function:* Name of the macro.

*Format:* Must conform to the rules for assembler language symbols.

*Default:* None. Specification optional.

```
ADDR={name      }
     {(register)}
```

*Function:* Identifies the first main storage address to be displayed.

*Format:* *name* is the symbolic name of the field from which main storage display should begin.

(*register*): decimal integer within parentheses, minimum 1, maximum 12.

*Default:* None. This parameter is required.

*Note:* The specified register contains the main storage address of the message or other area for which display is being requested.

```
CONV={YES}
     {NO }
```

*Function:* YES specifies that the system console will be given the opportunity of receiving more than the one main storage display specified in the macro, as well as the option of bypassing the last record accessed, as described in Chapter 4, "Displaying Main Storage from within an Application Program."

NO specifies that only the initial macro-specified display will be provided.

*Format:* YES or NO

*Default:* CONV=NO.

```
NAME={name      }
     {(register)}
```

*Function:* Supplies the identifier that precedes the display characters on the first print line.

*Format:* name: eight characters or less.

*(register):* decimal integer within parentheses, minimum 1, maximum 12

*Default:* The name of the current CSECT will be used to identify messages.

*Note:* If (*register*) is used, the specified number is of a register containing the address of an 8-byte field that includes the identifier name.

```
WORK={name       }
     {(register)}
```

*Function:* Specifies a 184-byte, doubleword-aligned work area.

*Format:* name: the name of the work area.

(*register*): decimal integer within parentheses, minimum 2, maximum 12.

*Default:* The work area is internally generated.

*Note:* If (*register*) is used, the specified register contains the address of the work area.

## Return Codes

One of the following return codes will be passed back to the application program in register 15 after the COREDSP macro executes.

| **Code** | **Meaning** |
|---|---|
| **X'00000000'** | Indicates that either CONV=NO has been coded or CONV=YES, has been coded and the operator has entered GO in response to COREDSP indicating current message should be processed. |
| **X'00000004'** | Indicates the operator responded NO to CORESP which indicates current message should not be processed. |

# DCB Macro (Input)

The input DCB macro:

- Defines an input data set for an application program
- Must be issued for each process queue addressed by the application program with GET or READ macros
- Specifies whether BSAM-compatible macros or QSAM-compatible macros are to transfer messages or records from the MCP to the application program
- Specifies the length in bytes of the application-program work area to which data is transferred from the MCP
- Specifies the length in bytes of buffers to be used in the MCP to transfer messages from the process queue to the application-program interface
- Specifies whether the application program is to handle entire messages or message portions called logical records

- Specifies the format and characteristics of records in the input data set
- Indicates the address of a routine to be given control when the end of a user-defined series of data records is reached
- Indicates the address of a routine to be given control when message overflow occurs.

Refer to Chapter 2 for details on using the (INPUT) DCB macro.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | DCB | `BLKSIZE=integer`<br>`,DDNAME=symbol`<br>`,DSORG=PS`<br>`,LRECL=integer`<br>`,MACRF={GM }`<br>`        {GL }`<br>`        {GLT}`<br>`        {GMT}`<br>`        {RP }`<br>`        {R  }`<br>`[,BUFL=integer]`<br>`[EODAD=address]`<br>`[,EXLST=address]`<br>`[OPTCD=[W][U][C]]`<br>`[,RECFM={GL } ]`<br>`         {V }`<br>`         {VB}`<br>`         {U }`<br>`[,STOP={QUICK} ]`<br>`       {FLUSH}`<br>`       {BOTH }`<br>`[,SYNAD=address]` |

*Note:* Other DCB operands may be coded, but only the ones detailed here are used by TCAM. Since these operands are all keyword operands, they may be coded in any order.

symbol

*Function:* Allows symbolic addressing of this instruction within a program; this symbol becomes the name of the TCAM data control block generated by the expansion of this macro.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this symbol is required.

BLKSIZE=integer

*Function:* Specifies the length (in bytes) of the application-program work area that this DCB controls.

*Format:* A decimal integer no smaller than the length of a record as specified by the LRECL operand of this DCB, and no larger than 32,760, or the APWAS operand specified on the INTRO macro.

*Default:* None. Specification of this operand is required either here or by the alternate source.

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition if it is provided at program execution by the DCB operand of the appropriate job control language data definition card.

The total length of all optional fields in your work area must also be included in this value. (See the OPTCD operand.) TCAM uses the BLKSIZE operand and LRECL parameters to determine the length of the record it will send to the work area in your program in response to each GET or READ macro.

For undefined-length work units, the value specified here will be dynamically overridden on a work-unit-by-work-unit basis by the *length* operand of each associated READ macro.

`BUFL=integer`

*Function:* Specifies the length, in bytes, for each application program buffer.

*Format: integer* is a decimal value no greater than 32,760.

*Default:* If the BUFL operand is omitted, the system acquires INTRO macro buffers with a length equal to the sum of the values specified in the KEYLEN and BLKSIZE operands. If the application program requires larger buffers, the BUFL operand must be specified.

*Alternate Source:* The BUFL operand can be specified in the DCB subparameter of a DD statement or by the application program before completion of the data control block exit routine.

`DDNAME=symbol`

*Function:* Specifies the symbolic label of the job control language data definition statement associated with this data control block.

*Format:* An arbitrarily assigned character string of up to eight characters unique to this program. It must be the same as the DD card label that defines this DCB to the operating system.

*Default:* None. Specification of this operand is required either here or by the alternate source.

*Alternate Source:* The value for this operand may be provided any time before this DCB is opened. See the DCB discussion in Chapter 2 for details on other ways that this value may be specified.

`DSORG=PS`

*Function:* Specifies that the data set identified by this data control block is organized in the physical sequential mode.

*Format:* PS

*Default:* None. Specification of this operand is required.

*Alternate Source:* None. This operand must be specified in the application program.

*Note:* This operand allows TCAM to achieve compatibility with either QSAM (GET/PUT) or BSAM (READ/WRITE).

`EODAD=address`

*Function:* Specifies the symbolic address of an open or closed user-written subroutine that is to be given control if TCAM recognizes a user-generated end-of-file indication in the header of a message during normal message processing.

*Format:* Must be identical to the symbolic name assigned to the routine that you wrote to handle an end-of-file condition. *address* must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this operand is optional unless the STOP operand was coded on this DCB.

*Alternate Source:* The value for this operand may be provided any time before the DCB created by this macro is opened. (See the DCB discussion in Chapter 2.)

*Note:* TCAM will take an exit to this routine when the next GET or CHECK macro is issued following a GET macro that transferred a work unit that ended with the end-of-file character. Time of entry to EODAD must be controlled by the user because of the real-time nature of the arrival of messages to a process queue for an application program. If an end-of-data character has not been specified in TCAM and a SETEOF macro is not issued in the MCP, you must provide some other means of determining end-of-data on input.

`EXLST=address`

*Function:* Specifies the symbolic address of a list of housekeeping or exit routines.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must conform to the rules for assembler language symbols.

*Default:* None. Specification of this operand is optional.

*Alternate Source:* The value for this operand may be provided any time before the DCB created by this macro is opened. (See the DCB discussion in Chapter 2.)

*Note:* You are responsible for coding all the routines addressed by this list. The list must start on a fullword boundary; its format and contents are shown and described in the *Data Management Services* manual. Each entry in the list must be a fullword made up of a control byte followed by a three-byte address of a user-written routine.

Only two entries in the list (those having control bytes of X'05' and X'0F') are meaningful for a TCAM input DCB. If the control byte is X'05', a DCB exit is taken. (See the *Data Management Services* manual for details.)

The routine with the control byte of 0F' is given control to initiate an MVS checkpoint of the application program. (See the section on coordinating MVS and TCAM checkpoints in Chapter 4.)

Upon entry to any of the routines specified by this exit list, the contents of registers 0 and 2 through 13 are the same as they were just before the associated GET or CHECK macro was executed. Register 1 contains the address of the input DCB created by this macro, and register 14 contains the return address of the application program. The just entered routine must save and restore the contents of registers 1 and 14. The contents of the user-defined save area must not be altered.

```
LRECL=integer
```

*Function:* Specifies the length (in bytes) of a work unit, or the length of a single logical record in your input data set plus all of the optional fields you have specified by the OPTCD operand on this macro.

*Format:* A decimal integer no larger than the APWAS operand specified on the INTRO macro.

*Default:* None. If RECFM = F has been specified on this DCB, this operand is required. Otherwise, specification is optional.

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition if it is provided at program execution by the DCB operand on the appropriate job control language data definition card. Specification here always takes preference over a DD card value.

*Note:* If RECFM = U is specified on this DCB, after each GET or READ macro the LRECL operand field in the data control block created by this macro is updated with the number of bytes of data that have been fetched. This total would include the record length, plus the length of all the optional fields preceding the record. (See the OPTCD operand of this macro.)

```
MACRF={GM }
      {GL }
      {GLT}
      {GMT}
      {RP }
      {R  }
```

*Function:* Specifies the type of MVS access method that will be used to retrieve records from the process queue defined by this DCB.

*Format:* GM, GL, GMT, GLT, R, or RP.

*Default:* None. Specification of this operand is required.

*Alternate Source:* None. This operand must be specified on a DCB macro in the application program.

*Note:* *G indicates GET for QSAM-compatibility; R indicates READ for BSAM-compatibility. GET may be specified in either move (M) or locate (L) mode.*

*T indicates that you intend to use the POINT macro in conjunction with the GET macro. P indicates that you intend to use the POINT macro in conjunction with the READ macro.*

*If locate (L) mode is specified with GET, TCAM obtains space for an application program work area by executing the GETMAIN macro instruction when the data set is opened. The space for the work area is obtained from the pageable area of main storage that is available to the application program. TCAM returns the address of the space to be used for the work area to the program in register 1 following execution of the first GET macro, and the program uses this same work area for all succeeding GETs until the program ends. See the section on "Dynamic Work-Area Definition" in Chapter 3.*

OPTCD=

*Function:* Specifies the types of optional TCAM control fields that you wish to precede the work unit in your program work area.

*Format:* W, WU, WC, WUC, U, UC, or C. (You may choose any combination of fields.)

*Default:* None. Specification of this operand is optional unless you require one or more option fields in your work area because of other operand specifications.

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition and provided at program execution by the DCB operand on the appropriate job control language data definition card.

*Note:* *W specifies that you wish an eight-byte field called the origin field to be allocated immediately preceding the work unit. This field is provided so TCAM may insert the symbolic name of the source of each message received by your program. TCAM places the name of the source, in EBCDIC, in the field left-justified and padded on the right with blanks. If W is coded but TCAM cannot determine the message source, the field is filled with eight character blanks (X'40').*

*U specifies that you intend to use message rather than record processing (see Chapter 3). If U is omitted, the work unit is assumed to be a record.*

*C specifies that you want a one-byte control field, called the position field, to be included in the work area. This field will be used by TCAM to indicate whether the work unit currently being read into the work area is the first, an intermediate, or the last segment of the message if message processing was specified; or whether a record* *delimiter has been detected in the data if record processing had been specified. When TCAM passes a work unit to your program, it fills the position field with a specific control byte depending on the contents of the work area. The control bytes that can be entered in this field are:*

| **Control Byte** | **Work Area Contents** |
|---|---|
| **X'40'** | *Intermediate portion of message* |
| **X'F1'** | *First portion of message* |
| **X'F2'** | *Last portion of message* |
| **X'F3'** | *An entire message* |

If you had specified RECFM = U (undefined-length work units), RECFM = V (variable-length), or RECFM = VB (variable-length blocked), and OPTCD = C or CW, the control byte will have one of the following meanings:

| **Control Byte** | **Work Area Contents** |
|---|---|
| **X'F4'** | Intermediate portion of message, end-or-record |
| **X'F5'** | First portion of message, end-of-record |
| **X'F6'** | Last portion of message, end-of-record |
| **X'F7'** | An entire message, end-of-record |

The control byte will have one of the four values X'F4' through X'F7' only if the end-of-record delimiter specified on the TPROCESS macro is the last byte of data in the work unit.

```
RECFM={F}
      {V}
      {B}
      {U}
```

*Function:* Specifies the format characteristics of the work units in the data set controlled by the DCB being created by this macro.

*Format:* F, V, VB, or U.

*Default:* RECFM = U.

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition if it is specified at program execution by the DCB operand on the appropriate job control language data definition card.

*Note:* RECFM = F specifies that the length of the work units are to be fixed. You must specify the length of each work unit obtained, plus the length of any optional field in the work area, in the LRECL field of this DCB macro. RECFM = F should be coded only when the number of bytes of data in a complete message will always be an exact multiple of the number of bytes specified by the LRECL operand. Otherwise, the last segment to be passed by TCAM will contain fewer bytes than the number specified in the LRECL operand, thus causing a read error.

If RECFM = V is coded on this DCB macro, TCAM places a four-byte prefix into the work area for each GET or READ macro. The first two bytes of the prefix will contain the binary sum of the length of the work unit plus four bytes (the length of the prefix). The second two bytes of the prefix will be binary zeros.

V specifies that the length of the work units in the data set that will be controlled by this DCB will be variable. Each work unit must be preceded in the work area by a standard four-byte SAM-prefix. (See the discussion in Chapter 3 on optional fields in the work area.)

If RECFM=VB and MACRF=R are coded on this DCB macro, TCAM places an eight-byte prefix into the work area for each READ. The first two bytes of the prefix will contain the binary sum of the length of the work unit plus eight bytes (the length of the prefix). The second two bytes will be binary zeros. The third two bytes will contain a binary number four less than that contained in the first two bytes. The final two bytes will be binary zeros.

VB specifies that the variable-length work units will be treated by TCAM as if they were blocked. Regardless, only one work unit is transferred to the work area per GET or READ macro. The work area must include an eight-byte SAM-prefix if MACRF=R and RECFM=VB are specified, and a four-byte prefix otherwise. If RECFM=VB is specified, the blocking factor must be one.

U specifies that the length of the work units you expect with each GET or READ macro is undefined. TCAM requires no prefix in the work area for this format. The length of each work unit passed to the work area is stored by TCAM in the LRECL field in the input data control block at the time the work unit is transferred.

```
STOP={QUICK}
     {FLUSH}
     {BOTH }
```

*Function:* Specifies the type of end-of-data processing action that you wish to be taken if a TCAM closedown is issued while this application program is executing.

*Format:* QUICK, FLUSH, or BOTH.

*Default:* None. Specification of this operand is optional, but if it is coded, EODAD must also be coded.

*Alternate Source:* The value for this operand may be provided any time before this DCB is opened. (See the DCB discussion in Chapter 2.)

*Note:* QUICK specifies that the EODAD exit of this DCB is to be taken on a quick closedown only. FLUSH specifies that the EODAD exit is to be taken on a flush closedown only. BOTH specifies that the EODAD exit is to be taken for either type of closedown.

```
SYNAD=address
```

*Function:* Specifies the symbolic address of an open or closed subroutine that you have coded. When record processing is used, this routine is to be given control if a work unit is received that is larger than the program work area and if OPTCD=C was not specified in this DCB.

*Format:* Must be identical to the symbolic name assigned to a routine in this region that you coded to handle this error condition. *address* must conform to the rules for assembler language symbols.

*Default:* None. Specification of this operand is optional.

*Alternate Source:* The value for this operand may be provided any time before this DCB is opened.

*Note:* For more information on SYNAD routines, see the input DCB macro discussion in Chapter 2.

### Return Codes

None.

## DCB Macro (Output)

The output DCB macro:

- Defines an output data set for an application program
- Must be issued for each process entry set up to receive messages or logical records from an application program
- Specifies whether QSAM-compatible macros or BSAM-compatible macros are to transfer messages or logical records from the application program to the MCP
- Specifies the format and characteristics of records in the data set
- Specifies the length of the MCP buffers used to receive messages from this application program
- Specifies the address of a routine to be given control when logical output errors occur
- Specifies the address of the problem-program exit list.

Refer to Chapter 2 for details on using the DCB (OUTPUT) macro.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | DCB | BLKSIZE=integer<br>,DDNAME=symbol<br>,DSORG=PS<br>,LRECL=integer<br>,MACRF={W }<br>{PM}<br>{PL}<br>[,BUFL=integer]<br>[,EXLST=address]<br>[OPTCD=[W][U][C]]<br>[,RECFM={F } ]<br>{V }<br>{VB}<br>{U }<br>[,SYNAD=address] |

*Note:* Other operands may be specified, but only the ones detailed here are used by TCAM. Since these operands are all keyword operands, they may be coded in any order.

`symbol`

*Function:* Allows symbolic addressing of this instruction within a program; this symbol becomes the name of the data control block generated by the expansion of this macro.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this symbol is required.

`BLKSIZE=integer`

*Function:* Specifies the length (in bytes) of the application-program work area that the DCB created by this macro controls.

*Format:* A decimal integer no larger than the APWAS operand specified on the INTRO macro.

*Default:* None. If MACRF=PL is specified on this DCB, this operand is required. Otherwise, specification is optional.

*Alternate Source:* This operand can be omitted from this DCB macro definition if it is provided at program execution by the DCB operand on the appropriate job control language data definition card.

*Note:* The length of any optional fields (see the OPTCD operand discussion) in the work area should be included in the value specified for this operand.

`BUFL=integer`

*Function:* Specifies the length in bytes of the buffers in the TCAM MCP that are to receive messages from this application program.

*Format:* A decimal integer equal to or greater than the value of the UNITSIZE operand on the MCP INTRO macro.

*Default:* None. Specification of this operand is optional.

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition if it is provided by the DCB operand on the appropriate job control language data definition card.

*Note:* If this operand is omitted here, the value specified in the BUFSIZE operand of the PCB macro in the MCP is used. Caution should be exercised when using this operand because the value specified here overrides the value specified in the MCP.

`DDNAME=symbol`

*Function:* Specifies the symbolic label of the job control language data definition statement associated with this data control block.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must be the same as the label specified on the DD card that defines this DCB to the operating system.

*Default:* None. Specification of this operand is required.

*Alternate Source:* The value for this operand may be provided any time before this DCB is opened. See the DCB discussion in Chapter 2 for details on how this value may be specified.

*Note:* If this operand is omitted here, it must be specified from an alternate source.

`DSORG=PS`

*Function:* Specifies that the data set organization of the output data set controlled by this DCB is physical sequential.

*Format:* PS

*Default:* None. Specification of this operand is required.

*Alternate Source:* None. This operand must be specified in the application program.

*Note:* This operand allows your TCAM program to achieve compatibility with either QSAM (GET/PUT) or BSAM (READ/WRITE).

`EXLST=address`

*Function:* Specifies the symbolic address of a list of exit routines.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must conform to the rules for assembler language symbols.

*Default:* None. Specification of this operand is optional.

*Alternate Source:* The value for this operand may be provided any time before the DCB created by this macro is opened. (See the DCB discussion in Chapter 2.)

*Note:* You are responsible for coding all of the routines addressed by this list. The list must start on a fullword boundary; its format and contents are shown in the *Data Management Services* manual. Each entry in the list must be a fullword made up of a control byte, followed by a three-byte address of a user-written routine.

Only two entries in the list (those having control bytes of X'05' and X'0F') are meaningful for a TCAM output DCB. If the control byte is X'05', a DCB exit is taken. (See the *Data Management Services* manual for details.)

If the control byte is X '0F', the routine is given control to initiate an MVS checkpoint of the application program. (See the section on coordinating MVS and TCAM checkpoints in Chapter 4.)

Upon entry to any of the routines specified by this exit list, the contents of registers 0 and 2 through 13 are the same as they were just before the PUT or CHECK macro was executed. Register 1 contains the address of the DCB created by this macro, and register 14 contains the return address for the application program. Your routine must save and restore the contents of registers 1 and 14. The contents of the user-defined save area must not be altered.

`LRECL=integer`

*Function:* Specifies the length (in bytes) of a work unit, or the length of a single logical record in your input data set plus all of the optional fields you have specified by the OPTCD operand on this macro.

*Format:* A decimal integer no larger than the APWAS operand specified on the INTRO macro.

*Default:* None. If RECFM = F is specified on this DCB, this operand is required. Otherwise, specification is optional.

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition if it is provided by the DCB operand on the appropriate job control language data definition card.

*Note:* *If RECFM = U is specified on this DCB, after each GET or READ macro the LRECL operand field in the data control block created by this macro is updated with the number of bytes of data that have been fetched. This total would include the record length, plus the length of all the optional fields preceding the record. (See the OPTCD operand of this macro.) This may be done with a DCBD macro as described in the Data Management Macro Instructions manual.*

*If you specify RECFM = U on this DCB and also code a value for the LRECL operand, the value specified by the length operand of the WRITE macro overrides the value specified here in the LRECL operand.*

```
MACRF={PL}
      {PM}
      {W }
```

*Function:* Specifies the MVS access method by which messages are to be transferred from your application program to the TCAM destination queue.

*Format:*. PL, PM, or W.

*Default:* None. Specification of this operand is required.

*Alternate Source:* None. This operand must be specified in the application program.

*Note: P specifies that your messages are to be transferred by QSAM PUT macros. W specifies that the messages are to be transferred by BSAM WRITE macros.*

*PUT may be specified in move (M) mode or locate (L) mode.*

*If locate (L) mode is specified for PUT, TCAM dynamically obtains a work area by issuing a GETMAIN macro instruction. When the first PUT macro in your program is executed, TCAM returns the address of the acquired work area to your program in register 1. The record is not actually written out in the data set until the next PUT macro instruction is issued. (See the section on "Dynamic Work-Area Definition" in Chapter 3.)*

```
OPTCD=
```

*Function:* Specifies the type(s) of optional TCAM control field(s) that you require in the program work area of your application program.

*Format:* W, WU, WC, WUC, U, UC, or C. (You may choose any combination of fields.)

*Default:* None. Specification of this operand is optional unless you require one or more optional fields in your work area because of other operand specifications.

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition and provided by the DCB operand on the appropriate job control language data definition card.

*Note: W specifies that you wish an eight-byte field called the destination field to be allocated immediately preceding the work unit. A destination field allows you to insert the symbolic name of the destination in each message your program generates. You must place the message destination name in EBCDIC, in the destination field, left-justified and padded on the right with blanks.*

*U specifies that the work unit is a message. If U is omitted, the work unit is assumed to be a record.*

*C specifies that you want a one-byte control field called the position field to be included in the working area. This field will be used to indicate whether the work unit being handled is the first segment, an intermediate segment, or the last segment of a message. When you load a work unit into the work area for output, you must also fill the position field with the appropriate byte depending on the contents of the work area. The control bytes that you can specify are:*

| **Control Byte** | **Work Area Contents** |
|---|---|
| **X'40'** | Intermediate portion of message |
| **X'F1'** | First portion of message |
| **X'F2'** | Last portion of message |
| **X'F3'** | An entire message |

```
RECFM={F }
      {V }
      {VB}
      {U }
```

*Function:* Specifies the format characteristics of the work units that will be sent to the output data set defined by this DCB.

*Format:* F, V, VB, or U.

*Default:* RECFM=U

*Alternate Source:* The value for this operand can be omitted from this DCB macro definition if it is provided by the DCB operand on the appropriate job control language data definition card.

*Note:* *F specifies fixed-length work units. Prior to issuing a PUT or WRITE macro, the length of the work unit, plus the length of any optional fields in the work area, must have been placed in the LRECL field of the associated output data control block.*

*V specifies that the work units are to be variable length. For both BSAM-compatible and QSAM-compatible requests, each work unit loaded into the work area must be prefaced by a standard SAM-prefix option field of four bytes. The length of the work unit must be provided by the setting of the prefix before issuing a PUT or WRITE macro.*

*VB specifies that the variable-length records should be treated as blocked by TCAM, although only one work unit is transferred to the MCP by each PUT or WRITE macro. The variable-length record work area includes a SAM-prefix of eight bytes if a MACRF=W is specified, and four bytes if otherwise.*

*U specifies undefined-length work units. TCAM requires no work-area prefix in this case. The sum of the length of the work unit, plus the length of any optional fields in the work area, must be placed in the LRECLfield of the DCB created by this macro prior to each PUT or WRITE macro, unless it is specified by the length operand of the WRITE macro.*

`SYNAD=address`

*Function:* Specifies the symbolic address of a routine that is to be given control if a logical output error occurs.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must conform to the rules for assembler language symbols and be the same as the symbolic name specified on your SYNAD routine.

*Default:* None. Specification of this operand is optional.

*Alternate Source:* The value for this operand may be provided any time before this DCB is opened. (See the DCB discussion in Chapter 2.)

*Note:* You are responsible for coding the SYNAD routine. For more information on SYNAD routines, see the SYNAD discussion in Chapter 2.

### Return Codes

None.

## DKJFND Macro

The DKJFND macro:

- Provides the TCAM application program with a dummy section (DSECT) containing the TCAM fixed header prefix fields; you can symbolically reference these fields when the DSECT is present.
- May be coded once in an application program.

After you have established addressability for the generated DSECT, you may reference any symbolic FHP field name.

The format of the DSECT generated by the DKJHFD macro is given in the *TCAM Program Reference Summary.*

In addition to the FHP DSECT, the DKJHFD macro may also generate a DSECT for the buffer prefix. Unlike the FHP, however, the buffer prefix is not passed in a message to the application program.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | DKJFND | [TCAM={<u>YES</u>} ]<br>{NO } |

`symbol`

*Function:* Specifies the name of the macro.

*Format:* Must conform to the rules for assembler language symbols.

*Default:* None. Specification optional.

```
TCAM={YES}
     {NO }
```

*Function:* Specifies whether the DSECT should contain the names of the fields in the TCAM buffer prefix.

*Format:* YES or NO

*Default:* TCAM=YES.

*Note:* NO should be used in an application program because the buffer prefix is not passed in the application program work area.

If NO is used, only the FHP fields are generated and the macro must be preceded immediately by the following assembler instruction:

symbol DSECT

## GET Macro

The GET macro:

- Obtains work units from the MCP for processing
- May be coded more than once in an application program

Refer to Chapter 3.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | GET | debname [,areaname] |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`areaname`

*Function:* Specifies a symbolic address of an application-program work area into which you wish the work unit to be placed.

*Format:* Must be identical to the label of your application-program work area. Register notation may be used.

*Default:* None. If move mode (GM or GMT) is specified in the MACRF operand of the input DCB macro associated with this GET, this operand is required. Otherwise, specification is optional.

*Note: If register notation is used, the register number must be enclosed in parentheses and the address of the work area must have been previously loaded into the register. Permissible registers are 1 through 12.*

*This operand may be omitted if MACRF=GL or GLT was specified in the input DCB macro. In this case, TCAM dynamically obtains a work area from pageable main storage by issuing a GETMAIN macro instruction when the input DCB is opened. After the first GET macro is issued, TCAM returns the address of the work area in register 1. TCAM transmits work units to your program in this same work area until the DCB is closed.*

`dcbname`

*Function:* Specifies the symbolic address of the input data control block associated with the process queue from which this instruction is to obtain work units.

*Format:* Must be identical to the label of the DCB macro defining the data set you wish to refer to. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number (1 through 12) must be enclosed in parentheses, and the address of the data control block must have been previously loaded into the register.

## Return Codes

One of the following return codes will be passed to your program in register 15 after a GET macro executes:

| **Code** | **Meaning** |
|---|---|
| **X'00000000'** | Normal completion; data is in work area. |
| **X'00000004'** | A SETEOF was executed in the MCP (no data in work area). A closedown is in progress; an EODAD exit was not taken. |
| **X'00000008'** | The work area overflowed; a SYNAD exit was not taken. |
| **X'0000000C'** | A message was not found or a message was incomplete after a POINT/GET macro was executed (Retrieve mode). |
| **X'00000010'** | The blocksize is zero or invalid (larger than the value specified on the APWAS operand of the INTRO macro) or LRECL=0 and RECFM=F was specified. |
| **X'00000080'** | The GET macro cannot execute because the MCP is not active. |

## MCOUNT Macro

The MCOUNT macro:

- Returns, in register 1, the number of complete messages on the input queue
- Can be issued in an application program before a GET or READ macro to determine how many messages are queued for the application program.

Refer to Chapter 4 for details on using the MCOUNT macro.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `MCOUNT` | `DCB=dcbname` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`DCB=dcbname`

*Function:* Specifies the symbolic name of the data control block that defines the message queue whose messages you wish to count.

*Format:* Must be identical to the label specified on the DCB macro that controls the process queue for which you wish to count messages. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* *If register notation is used, any of general registers 2 through 12 may be used. The register must be loaded with the address of the DCB. The specified register number must be enclosed in parentheses.*

*If dcbname defines an output data control block, a count of zero is put in register 1 by TCAM.*

## Return Codes

| Code | Meaning |
|---|---|
| **X'00000000'** | MCOUNT executed successfully. |
| **X'00000004'** | MCOUNT did not execute because TCAM is not in the system. |
| **X'00000008'** | MCOUNT did not execute because the specified DCB does not define a data set for TCAM messages. |
| **X'0000000C'** | MCOUNT did not execute because *dcbname* named an output DCB. |

## MCPCLOSE Macro

The MCPCLOSE macro:

- Initiates closedown of TCAM
- Is optional in an application program.

MCPCLOSE may be issued in an application program to initiate TCAM closedown. For successful execution of MCPCLOSE, the issuing application program must meet one of the following conditions:

- Application program has an open TCAM DCB.
- Attacher of the application program has an open TCAM DCB.
- Refer to Chapter 2 for stopping the MCPCLOSE macro.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | MCPCLOSE | {QUICK}<br>{<u>FLUSH</u>}<br>[{,PASSWRD=chars}] |

symbol

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

{QUICK}
{<u>FLUSH</u>}

*Function:* Specifies the type of MCP closedown that you wish to occur (see *Notes*).

*Format:* QUICK or FLUSH

*Default:* FLUSH

*Note:* *QUICK specifies that message traffic is to cease upon completion of the reception or transmission by the MCP of any message currently in progress. Messages queued for destinations are not transmitted unless the MCP is subsequently restarted (warm start).*

*FLUSH specifies that input message traffic to TCAM is to cease immediately. The network will be closed down after completion of processing of all messages currently in the MCP. All messages completely queued for the destinations are transmitted, but messages in a middle-of-message condition are not completed or transmitted.*

`PASSWRD=chars`

*Function:* Specifies the TCAM protection password that enables only qualified application programs to issue this macro.

*Format:* One to eight nonblank characters conforming to the rules for assembler language symbols. *chars* must be identical to the value coded on the PASSWRD operand of the INTRO macro in the MCP.

*Default:* None. If the PASSWRD operand is specified on the INTRO macro in the MCP, this operand is required. Otherwise, this operand should not be specified.

## Return Codes

One of the following return codes will be passed back to your program in register 15 after the MCPCLOSE macro executes:

| **Code** | **Meaning** |
|---|---|
| **X'00000000'** | MCPCLOSE executed successfully. |
| **X'0000000C'** | MCPCLOSE did not execute because either TCAM is not in the system, or because the application program did not meet one of the two conditions listed at the beginning of the discussion of the MCPCLOSE macro. |
| **X'00000014'** | MCPCLOSE did not execute because either an invalid protection password is specified in the PASSWRD operand, or the PASSWRD operand is not specified and should be because a password was specified on the INTRO macro in the MCP. |

## MRELEASE Macro

The MRELEASE macro:

- Releases messages queued for a destination
- Reactivates a destination made inactive by a HOLD macro or an Intercept a Station operator command.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `MRELEASE` | `statname`<br>`[,PASSWRD=chars` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`PASSWRD=chars`

*Function* Specifies the specific protection password that enables only qualified application programs to issue this macro.

*Format:* One to eight nonblank characters conforming to the rules for assembler language symbols. *chars* must be identical to the value coded on the PASSWRD operand of the INTRO macro in the MCP.

*Default:* None. If the PASSWRD operand is specified on the INTRO macro in the MCP, this operand is required. Otherwise, this operand should not be specified.

`statname`

*Function:* Specifies the symbolic name of the program or external LU for which you wish to release all the queued messages.

*Format:* Must be the same as the symbolic name of the external LU entry in the terminal-name table related to this external LU or program. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the address of the symbolic name of the program or external LU must be placed in a general register. The name must be left-justified and padded with blanks to the length of eight bytes. Any of these registers 2 through 12 may be used. The register number must be enclosed in parentheses.

### Return Codes

One of the following return codes will be passed back to your program in register 15 after the MRELEASE macro executes:

| Code | Meaning |
|---|---|
| **X'00000000'** | MRELEASE executed successfully. |
| **X'00000004'** | MRELEASE did not execute because the external LU or program is already receiving its queued messages. |
| | MRELEASE did not execute because either the protection password specified in the PASSWRD operand does not match the protection password specified by the PASSWRD operand of the INTRO macro, or a protection password is not specified and it must be because password protection is specified in the MCP. Code the PASSWRD operand value exactly as it is coded in the PASSWRD operand of the INTRO macro in the MCP. |
| | MRELEASE did not execute because *statname* is not a single entry in the terminal table, there is no HOLD macro coded in the MCP, or the external LU specified uses main-storage-only message queues. |
| **X'0000000C'** | MRELEASE did not execute because TCAM is not in the system or (in a system which allows multiple TCAMs) there is no open DCB in your application program. |
| **X'00000020'** | MRELEASE did not execute because an invalid external LU or program is specified in the *statname* field. |

## OPEN Macro

The OPEN macro for the application program:

- Completes initialization and activation of the input and output data sets for the application program
- Is required to activate each data set represented by an input or output DCB macro
- Tests for proper authorization.

Refer to Chapter 2 for details on the OPEN macro.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | OPEN | (dcbname,,...) MF={L } {(E,listname)} |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. If MF=L is coded as an operand of this macro, this label is required. Otherwise, specification is optional.

*Note:* If MF=L is specified on this macro, this label becomes the name of the parameter list generated by this macro. This label must be specified as *listname* in a corresponding MF=E form of this macro.

`(dcbname,,...)`

*Function:* Specifies the symbolic names of all the data control blocks that you wish to be opened with this instruction. The ellipsis indicates that you may specify multiple *dcbname* parameters.

*Format:* Each *dcbname* must be identical to the symbolic name of the DCB that controls each of the data sets being opened. Register notation may be used.

*Default:* None. Specification of this operand is required except when a list name is specified in the execute form of this macro.

*Note:* If register notation is used, the specified register should contain the symbolic name of the data control block for the data set being opened. Any of the registers 2 through 12 may be used. The register number must be enclosed in parentheses. If multiple *dcbname* parameters are specified, they must be separated by double commas. For example, for three *dcbnames*, you would code (DCBA,,DCBB,,DCBC) and for register notation, you would code ((3),,(4),,(5)).

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

*Function:* Specifies whether this instruction is to be an executable instruction (MF=E), or a list instruction (MF=L) that must be referred to by another OPEN macro instruction. See *Data Management Macro Instructions* for a discussion on the list and execute forms.

*Format:* L or (E,*listname*), where *listname* must conform to the rules for assembler language symbols.

*Default:* MF=E

*Note:* MF=L allows you to create a parameter list. No executable code is generated. You must specify this form of the OPEN macro among your program constants. The parameters in the list are not used until the application program issues an OPEN macro with an MF=(E, *listname)* operand that refers to the list. The label specified in the name field of the list form of OPEN becomes the name assigned to the parameter list. This label must be coded as the *listname* value for an execute form of the instruction.

MF=(E,*listname)* executes the OPEN routine, using the parameter list referred to by *listname*. Any parameters specified in a macro having the MF=(E,*listname)* operand override corresponding parameters in the list.

## Return Codes

None.

## POINT Macro

The POINT macro:

- Permits message identification, causing the next GET or READ macro to retrieve the desired messages, (i.e. messages destined for the terminal or TPROCESS entry or messages for which the terminal or TPROCESS entry is a source).

Refer to Chapter 3 for details on the POINT macro.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `POINT` | `dcbname`<br>`,address` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`address`

*Function:* Specifies the symbolic address of a field in your program that contains control information necessary for the execution of this macro. See Notes.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must conform to the rules for assembler language symbols. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note: address is the symbolic address of an eleven-byte block in your program containing three fields:*

- *The first is an eight-byte field containing the external LU name, left-justified and padded with character blanks (X'40'). This field must be initialized with a valid external LU name or name of a TPROCESS entry. The retrieval function (GET or READ) then retrieves messages from this external LU's destination queue or the queue associated with the TPROCESS entry.*
- *The next byte contains a character which identifies the type of retrieval desired. I (X'C9') for retrieval by input sequence numbers. S (X'E2') for retrieval by output sequence numbers of sent (serviced) messages, (MVS only). O (X'D6') for retrieval by output sequence numbers of both sent and unsent messages. (X'40') to request termination of the retrieval function.*
- *The last is a two-byte message sequence number, specified in binary and right-justified with leading zeros. If multiple retrieval is desired, the first byte must be X'80'. The sequence number is the number of the message that message retrieval starts with.*

*If register notation is used, the address of this area must have been previously loaded into one of the registers 2 through 12. The register number must be enclosed in parentheses.*

dcbname

*Function:* Specifies the label of the data control block in your application program that defines the logical data set to which you will issue a subsequent GET or READ macro associated with this POINT macro.

*Format:* Must be identical with the label of the data control block that controls the data set that you wish to access. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the address of the data control block must have been previously loaded into one of the general registers 2 through 12. The register number must be enclosed in parentheses.

## Return Codes

Registers that may be altered during execution of the POINT macro are 0, 1, 14, and 15. One of the following return codes is passed back to your application program in register 15 after the POINT macro executes:

| Code | Meaning |
|---|---|
| **X'00000000'** | POINT executed successfully. |
| **X'00000004'** | POINT did not execute because the message sequence number specified is invalid for this queue. This return code is only returned when the specified sequence number is 0. For more information, see the explanation of completion code X'50000000' in the description of the CHECK macro. |
| **X'00000008'** | POINT did not execute because the destination name specified is not a valid entry in the terminal-name table. |
| **X'0000000C'** | POINT did not execute because either: ● The specified destination queue is not located in a data set residing on disk ● You tried to specify an input process entry as the message source (that is, you specified *I* in conjunction with the name of a GET process entry) ● You tried to specify an output process entry as the message destination (that is, you specified *O* in conjunction with the name of a PUT process entry) ● DISK=NO was specified on the INTRO macro instruction in the MCP |
| **X'00000010'** | POINT request to terminate a retrieval did not execute because a retrieval operation was not in progress. |
| **X'00000014'** | POINT did not execute because the POINT parameter list was altered while a multiple retrieval operation was in progress. For example, an attempt to start a retrieval operation was made while a retrieval operation was in progress. **Programmer Response:** The original retrieval operation must be ended. Issuing a GET after receiving this return code will have unpredictable results. Retrieval may be ended by issuing a POINT blank followed by a GET or by closing and reopening the DCB. |

## PUT Macro

The PUT macro:

- Returns work units to the MCP after processing
- May be specified more than once in an application program

Refer to Chapter 3 for details on the PUT macro.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `PUT` | `dcbname`<br>`[,areaname]` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`areaname`

*Function:* Specifies the symbolic address of a work area in your program from which you wish a work unit to be transmitted.

*Format:* Must be identical to the symbolic name specified for your application-program work area. Register notation may be used.

*Default:* None. If move mode (MACRF=PM) is specified on the output DCB macro specified by the *dcbname* operand of this macro, this operand is required. Otherwise, specification is optional.

Note: *If register notation is used, the register number must be enclosed within parentheses and the address of the work area must have been previously loaded into that register. Any one of the general registers 0 or 2 through 12 may be used.*

*If MACRF=PL is specified on the referenced DCB macro, this operand should be omitted. In this case, the address of a work area into which your program must place the work unit for it to be transferred to the MCP is returned in register 1 by the first PUT macro referring to this DCB. The first PUT macro does not transfer any data; it only provides the address of a work area in your area of addressability. The second and all subsequent PUT macros will cause transfer of data from the address specified after the first PUT macro. You are therefore responsible for loading the work unit that you wish transferred into that work area. For more information on locate mode, see "Dynamic Work Area Definition" in Chapter 3.*

dcbname

*Function:* Specifies the label of the data control block for the output data set to which you wish to transmit an application program work unit.

*Format:* Must be identical to the label of the output DCB controlling the data set you wish to transmit a work unit to.

*Default:* None. Specification of this operand is required.

*Note:* The QNAME operand of the job control language DD statement for this DCB names a process entry in a terminal table in TCAM that is coded especially to receive messages from this application program.

Do not try to execute a PUT or WRITE macro in an application program if a PUT or WRITE macro is currently executing and indirectly referring to (by the TPROCESS entry) the same process control block (PCB). This condition could occur if two subtasks of the same application program with a single PCB tried to execute a PUT or WRITE macro. If for some reason the MCP must *wait* before the first operation can be completely processed, the second subtask of the same application program could gain control and try to execute a PUT or WRITE macro. As a rule, the MCP would be forced to *wait* only if there was a buffer shortage or if the message being processed was an operator control message that required a long time to process.

To guard against this condition, TCAM returns an indication. If an attempt is made to execute a PUT macro, TCAM will return an error indication X'10' in register 15. For a WRITE macro, the DECB will contain a completion code of X'5C000000'. Unlike other PUT/WRITE errors, the user is not required to close down the DCB affected.

If more than one subtask in the same application program includes PUT or WRITE macros, the possibility of this type of error can be eliminated by use of the ENQ and DEQ macros. ENQ can be coded before each PUT or WRITE, and DEQ can be coded after each PUT or WRITE macro. The resource must be a name that symbolizes the PCB.

If register notation is used, the register number specified must be enclosed in parentheses, and the address of the data control block must have been loaded previously into a register (1 through 12).

## Return Codes

One of the following return codes will be passed back to your application program register 15 after the PUT macro executes:

| Code | Meaning |
|---|---|
| **X'00000000'** | PUT executed successfully. |
| **X'00000004'** | PUT cannot execute because a quick closedown of the MCP has begun. |

| | |
|---|---|
| **X'00000008'** | PUT cannot execute because the position-code field preceding the work unit contains an invalid or out-of-sequence value. |
| **X'0000000C'** | PUT cannot execute because the destination field contains a name that is not a valid entry in the terminal-name table. |
| **X'00000010'** | PUT cannot execute because TCAM has no buffers available due to heavy message traffic. PUT cannot execute because simultaneous PUTS were issued against the same PCB. A PUT should be reissued. |
| **X'00000014'** | PUT cannot execute because an invalid BLKSIZE operand was specified on the output DCB macro. The DCB macro BLKSIZE operand value was greater than the INTRO macro APWAS operand value. The BLKSIZE operand value must be equal to or greater than the work unit length, and less than or equal to 32,760. |

*Note:* When a PUT or WRITE/CHECK is issued, with the destination's name in the work area, TCAM verifies that the named destination is in the terminal-name table. Verification that the named destination is capable of receiving data is not done until the message passes through the INHDR (in header) subgroup of the AMH (application program message handler) in the TCAM MCP. Therefore, if the name exists in the terminal name table, a return code will be received on the PUT or CHECK macros even though the message cannot go to the named destination. The AMH must check the return code from its FORWARD macro to ensure that the destination is valid. This return code will be non-zero if the named external LU is not capable of receiving output from TCAM.

## QCOPY Macro

The QCOPY macro:

- When LIMIT is specified, displays an external LU QCB
- Displays all QCBs for external LUs having some threshold number of messages queued when LIMIT is specified
- Is optional in a TCAM application program

Refer to Chapter 4 for details on QCOPY macro.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `QCOPY` | `termname`<br>`,areaname`<br>`[,LIMIT={integer    }]`<br>`         {(register)}` |

symbol

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

areaname

*Function:* Specifies the symbolic name of the application-program work area into which you wish the contents of the designated QCB to be placed.

*Format:* Must be identical to the symbolic name that you have specified for the work area you wish to copy the QCB into. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number must be enclosed in parentheses and the register must have been previously loaded with the address of the work area. Permissible registers are 2 through 12.

```
LIMIT={integer    }
      {(register)}
```

*Function:* Allows you to specify a minimum number of messages that must be queued for this *termname* before TCAM will select the entry for copying.

*Format:* A decimal integer with a minimum value of 1 and a maximum of 4095. Register notation may be used.

*Default:* None. Specification of this operand is optional.

*Note:* If register notation is used, the register number must be enclosed within parentheses. The limit that you wish to specify must have been loaded into the register in binary. Registers 2 through 12 may be used. If this operand is not coded, the QCOPY macro will copy the QCB for the designated external LU name. If this operand is coded, and some number of messages less than that specified on this operand is actually on the queue, TCAM will increment through the terminal-name table for an external LU name subsequent to the one specified in *termname* with at least the LIMIT number of messages in its queue. The name of this external LU will be returned by TCAM to the *termname* field and its QCB will be copied.

`termname`

*Function:* Specifies the symbolic name of the terminal-name table entry whose QCB is to be displayed, or the name of the terminal-name table entry you wish to start with.

*Format:* Must be identical to the symbolic name of the TERMINAL macro that defines the terminal-name entry for the queue control block you wish to copy. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* The name must be left justified and padded with blanks to the length of the longest external LU name in the table. If register notation is used, the register number must be enclosed within parentheses, and the register must contain the address of a field containing the name of the entry. Permissible registers are 2 through 12.

## Return Codes

One of the following return codes is passed back to your program in register 15 after the QCOPY macro executes:

| **Code** | **Meaning** |
|---|---|
| **X'00000000'** | QCOPY executed successfully. |
| **X'00000004'** | QCOPY did not execute because the resources named by *termname* does not have a QCB. |
| **X'00000008'** | QCOPY did not execute because TCAM is not in the system or in a system which allows multiple TCAMs. There is no open DCB in your application program. |
| **X'0000000C'** | QCOPY did not execute because no TCAM application program DCB was open; at least one input or output DCB must be open for this macro to execute. |
| **X'00000018'** | QCOPY did not execute because the interface work area is too small. |
| **X'00000020'** | QCOPY did not execute because the resources named by *termname* were not found in the terminal name table. |

# QRESET Macro

The QRESET macro:

- Resends messages to an output device whose message queue resides on reusable or nonreusable disk
- May alter registers 0, 1, 2, and 15 during the execution of the QRESET function routines.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | QRESET | dcbname<br>,MAX=integer |

symbol

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

dcbname

*Function:* Specifies the name of some opened DCB in the application program except the DCB for the queue you wish to reset.

*Format:* Must be identical with the name of a DCB that is defined in the application program. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* *If you are attempting to reset an application-program queue, the DCB for that particular queue must be closed when the QRESET macro executes. Since at least one DCB in the program must be open for the QRESET macro to execute, one of the other DCBs in the program will have to be open. If register notation is used, the address of the DCB must have been previously loaded in one of the registers 3 through 12. The register number must be coded with parentheses.*

*The name of the queue you wish to be reset must be specified in the following format. The address of this ten-byte field must be loaded into register 2 before the QRESET macro executes.*

| Byte | Format | Contents |
|---|---|---|
| 0-7 | Character (left-justified) and padded with blanks) | Queue name as specified on the DCB defining that queue. |
| 8-9 | Hexadecimal | Sequence number-maximum value is decimal or 9999 (hex 270F). |

MAX=integer

*Function:* Allows you to specify the maximum number of messages that you wish to be re-sent from the queue specified in the ten-byte control field referred to by register 2. (See the QRESET macro discussion in Chapter 3).

*Format:* A decimal value greater than zero and less than or equal to 25. The number must be equal to or greater than the difference between the current SEQOUT number and the QRESET sequence number. If more than 25 messages are to be reset then the QRESET macro may be reissued, provided a return code of zero was received on the previous request.

*Default:* None. Specification of this operand is required.

## Return Codes

One of the following return codes is passed back to your program in register 15 after the QRESET macro executes:

| Code | Meaning |
|---|---|
| **X'00000000'** | QRESET has executed successfully. The terminal sequence out field is the last message reset. |
| **X'00000004'** | QRESET has been unable to mark some of the range of output sequence numbers as unserviced. Register 2 contains the count of messages successfully processed by QRESET. The terminal sequence out field is the last message reset. |
| **X'00000008'** | Reusable disk reorganization is in progress. Register 2 contains the count of messages successfully processed by QRESET prior to the start of reorganization. |
| **X'0000000C'** | QRESET did not execute because the requested output sequence number is invalid for one of the following reasons: ● It is higher than the last sequence number received at the specified output device ● The specified maximum value is less than the difference between the current SEQOUT number and the QRESET sequence number ● It is zero ● It is not marked serviced. |
| **X'00000010'** | QRESET cannot be issued from this application program because QBACK=YES was not specified on the TPROCESS macro, or this request was for an open application program DCB. The APWAS operand of the INTRO macro does not specify 440 or greater. |
| **X'00000014'** | QRESET did not execute because either the specified output device or the process entry name is invalid, or the device is currently being held (intercepted). |
| **X'00000018'** | QRESET did not execute because either an error condition regarding the type of queuing has been detected, or there are no messages queued for the specified external LU. |

| | |
|---|---|
| **X'0000001C'** | The queue reset function has been prematurely terminated because one of the requested headers may not be available due to reusable disk reorganization. Any complete messages found up to this point have been marked unserviced, but will not be scheduled to be re-sent. |

## READ Macro

The READ macro:

- Requests TCAM to transfer a work unit from the MCP to a designated work area in the application program
- May be coded more than once in an application program.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `READ` | `decbname`<br>`,SF`<br>`,dcbname`<br>`,area`<br>`{,length}`<br>`{'S'    }`<br>`[MF={L          }]`<br>`   {(E,listname)}` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`areaname`

*Function:* Specifies the symbolic name of the work in your program into which the incoming work unit is to be placed.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must also conform to the rules for assembler language symbols. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* The work area must be aligned on a fullword boundary and must be large enough to hold the largest work unit your program expects. If register notation is used, the address of the work area must have been previously loaded into a register, and the register number must be enclosed within parentheses. You may use any on the general registers 2 through 12.

```
dcbname
```

*Function:* Specifies the label of the data control block defining the input data set from which this READ macro will acquire a work unit.

*Format:* Must be identical to the name specified as the label of the DCB macro for the data set being read. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the address of the data control block must have been previously loaded into one of the general registers 2 through 12, and the register number must be enclosed within parentheses.

```
decbname
```

*Function:* Specifies a symbolic name to be assigned to the data event control block (DECB) created as part of this macro expansion. This symbol will be referred to by the CHECK macro.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this operand is required.

```
{length}
{'S'   }
```

*Function:* Specifies the length in bytes of the work unit, plus any optional fields, that are to be read into the work area.

*Format:* A decimal integer equal to 32,760 or no larger than APWAS operand specified on the INTRO macro.

*Default:* None. Specification of this operand is required only if RECFM=U is coded on the DCB associated with this READ macro.

*Note:* This operand should only be coded for undefined-length (RECFM=U) work units; it is ignored for fixed-length (RECFM=F) and variable-length (RECFM=V) work units. If 'S' is coded, and an undefined-length work unit is to be processed, the number of bytes to be read is taken from the LRECL operand of the data control block specified by *dcbname*. Note that S is enclosed within single quotes.

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

*Function:* Specifies whether this instruction is to be an executable instruction (MF=E), or a list instruction (MF=L) that must be referred to by another READ macro instruction. See the *Data Management Instructions* manual for a discussion of the list and execute forms.

*Format:* L or (E, *listname)* where *listname* is any arbitrarily assigned character sequence of up to eight characters that occurs as the label of another READ macro in this program. The other READ macro must define a list format (MF=L) macro.

*Default:* MF=E

*Note:* *MF=L creates a parameter list based on the other READ macro's operands. No executable code is generated. You must specify the list form of this READ macro among your program constants. The parameters in the list are not used until the application program issues a READ macro with an MF=(E, listname) operand that refers to the list. The label specified in the name field of the list form becomes the name assigned to the parameter list.*

*MF=(E, listname) executes the READ macro, using the parameter list referred to by listname. This list was created by a macro having the MF=L operand specified. Parameters specified in a macro having the MF=(E, listname) operand, override corresponding parameters in the list form of the macro.*

SF

*Function:* This operand establishes BSAM compatibility. It also specifies that any data set being read will be read sequentially.

*Format:* SF

*Default:* None. Specification of this operand is required.

### Return Codes

No return codes are returned in register 15. Completion codes from execution of the READ macro are placed in the event control block of the DECB for this macro. (See the preceding CHECK macro discussion in this chapter.)

## TCBINCNV Macro

The TCBINCNV macro instruction:

- Converts a halfword binary number to the equivalent printable decimal value expressed as a 5-byte EBCDIC number with leading zeros (if any) replaced by the EBCDIC blank character (X'40')
- Converts a halfword binary number to the equivalent hexadecimal value expressed as a 4-byte EBCDIC field
- Converts an EBCDIC decimal number (up to 5 bytes long) to the equivalent halfword binary number.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | TCBINCV | CONVTO= {<u>DEC</u>} { HEX } { BIN } ,INPUT= {(reg)} {name } ,OUTPUT= {( reg)} {name } |

symbol

*Function:* Name of the macro.

*Format:* Must conform to the rules for assembler language symbols.

*Default:* None. Specification optional

```
CONVTO={DEC}
       {HEX}
       {BIN}
```

*Function:* Specifies type of conversion to be performed.

*Format:* DEC, HEX, or BIN

*Default:* CONVTO=DEC

*Note:* *DEC converts a halfword binary number to the equivalent decimal value expressed as a 5-byte EBCDIC number, with any leading zeros replaced by EBCDIC blank characters (X'40'), in the output area specified by the user.*

*HEX converts a halfword binary number to the equivalent hexadecimal value expressed as a 4-byte EBCDIC field in the output area specified by the user.*

*BIN converts an EBCDIC decimal numeric field to the equivalent halfword binary number.*

```
INPUT={(reg)}
      {name }
```

*Function:* Provides the address of the field containing the input value.

*Format:* *(reg)* is a general register, 2 through 12, containing, if CONVTO=DEC or HEX, the address of the halfword-aligned area for binary input or, if CONVTO=BIN, an address pointer to the first EBCDIC decimal character of an input string to be converted from decimal to binary.

*reg* must be enclosed in parenthesis and can be specified as an explicit decimal integer from 2 through 12, or a symbol that has previously been equated to a decimal value from 2 through 12.

*name* is the name of a halfword-aligned area for binary input (if CONVTO=DEC or HEX) or the name of an area containing the first character of the LEBCDIC decimal number (if CONVTO=BIN).

*Default:* None. Specification required

*Note:* If CONVTO=BIN, the input decimal number to be converted is considered to be delimited by the first non-numeric character found, or by a maximum of 5 characters, whichever occurs first.

```
OUTPUT={(reg)}
       {name }
```

*Function:* Provides the address of an area that will contain the converted value.

*Format:* *(reg)* is a general register 2 through 12, containing the beginning address of the area where the converted output is to be placed. *(reg)* must be enclosed in parentheses and can be specified as an explicit decimal integer from 2 through 12. It is the user's responsibility to assign this operand a value that falls within the prescribed limits. *name* for binary to decimal conversion is the name of a field 5 bytes long.

*name* for binary to hex conversion is the name of a field 4 bytes long. *name* for a decimal to binary conversion is the name of a halfword-aligned area.

*Default:* None. Specification required.

## Return Codes

None.

## TCHNG Macro

The TCHNG macro:

- Places specified data in a terminal-table entry and its associated option fields
- Is optional in a TCAM application program.

Refer to Chapter 4 for details on TCHNG macro.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | TCHNG | termname<br>,areaname<br>[,PASSWRD=chars] |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`areaname`

*Function:* Specifies the symbolic name of the work area in your program from which TCAM is to obtain the information to move into the terminal table.

*Format:* Must be identical to the symbolic name that you specified for the work area. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note: The first byte of data in the work area must be the terminal table list status byte. The work area should be at least as long as the longest terminal-table entry that will ever be changed. When the entry is shorter than the work area, you must pad to the right with blanks to fill out the work area.*

*If register notation is used, the register number must be enclosed within parentheses, and the register must contain the address of the work area. Permissible registers are 2 through 12.*

*You are responsible for the integrity of the new entry. The entry must contain all necessary information and be in the proper form for successful operation of TCAM. See the description of the terminal-table entries in the TCAM Installation Guide.*

`PASSWRD=chars`

*Function:* Specifies the specific TCAM protection password that enables only qualified application programs to issue this macro.

*Format:* One to eight nonblank characters conforming to the rules for assembler language symbols. This value must be identical to the value coded on the PASSWRD operand of an INTRO macro in the MCP.

*Default:* None. If the PASSWRD operand is specified on the INTRO macro in the MCP, this operand is required. Otherwise, this operand should not be specified.

*Note:* If the character string specified in this operand does not match that specified in the INTRO macro, or if this operand is not coded but PASSWRD was coded on the INTRO macro, the TCHNG macro is ignored.

`termname`

*Function:* Specifies the symbolic name of the terminal-table entry whose contents are to be replaced by the contents of the work area named in the *areaname* operand.

*Format:* Must be identical to the symbolic name of the TERMINAL macro or TPROCESS macro that defines this entry in the terminal table. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number must be enclosed within parentheses, and the register must contain the address of a field containing the name of the terminal-table entry, left-justified and padded with blanks. The field must be at least as long as the longest external LU or program name in the terminal-name table, with a maximum of eight characters. Permissible registers are 2 through 12.

## Return Codes

One of the following return codes is passed back to your program in register 15 after the TCHNG macro executes:

| **Code** | **Meaning** |
|---|---|
| **X'00000000'** | TCHNG executed successfully.<br><br>*Note:* If operator mask prohibits changes, no change will occur. |
| **X'00000008'** | TCHNG did not execute because TCAM is not in the system or (in a system which allows multiple TCAMs) there is no open DCB in your application program. |
| **X'0000000C'** | TCHNG did not execute because no TCAM application program DCB is open; at least one input or output DCB must be open in order for this macro to execute. |
| **X'00000014'** | TCHNG did not execute because either (a) an invalid protection password is specified as the PASSWRD value of the TCHNG macro, or (b) the PASSWRD operand is not specified in the TCHNG macro. (PASSWRD must be specified because the INTRO macro specifies a protection password; code this operand exactly as it is coded in the INTRO macro). |
| **X'00000020'** | TCHNG did not execute because an invalid external LU name is specified in the *termname* field (that is, no such entry exists in the terminal-name table). |

| | |
|---|---|
| **X'00000040'** | TCHNG did not execute because the device dependent field was incorrectly changed by one of the following: ● A change was attempted on TRMDEVFL in order to add or delete a new field. ● A change was attempted on the length field of a device dependent field. |
| **X'00000044'** | TCHNG did not execute because it was attempting to add, delete, or change the length of an option field. |

## TCOPY Macro

The TCOPY macro:

- Permits examination of the contents of a terminal-table entry and its associated option fields
- Is optional in a TCAM application program
- Cannot be used with a GROUP macro.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `TCOPY` | `statname` `,areaname` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`areaname`

*Function:* Specifies the symbolic name of the application-program work area into which the terminal-table entry and its associated option fields are to be placed.

*Format:* Must be identical to the symbolic name that you specified for the program work area that you wish to copy the terminal-table entry into. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number must be enclosed within parentheses. The register must previously have been loaded with the address of the work area. Permissible registers are 0, and 2 through 12.

The work area must be aligned on a fullword boundary and be large enough to hold the longest terminal-table entry you expect to receive.

`statname`

*Function:* Specifies the symbolic name of the TERMINAL or TPROCESS entry whose queue contents are to be moved to your program work area.

*Format:* Must be the same as the symbolic name specified for the external LU or program in the MCP terminal-name table. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number must be enclosed within parentheses, and the register must have been previously loaded with the address of a field containing the entry name as specified on the TERMINAL or TPROCESS macro defining the entry. The name must be left-justified and padded with blanks to the length of the longest name in the terminal-name table. Permissible registers are 0, and 2 through 12.

## Return Codes

One of the following return codes is passed back to your program in register 15 after the TCOPY macro executes:

| Code | Meaning |
|---|---|
| **X'00000000'** | TCOPY executed successfully. |
| **X'00000008'** | TCOPY did not execute because TCAM is not in the system or (in a system which allows multiple TCAMs) there is no open DCB in your application program. |
| **X'0000000C'** | TCOPY did not execute because no TCAM application-program DCB is open; at least one input or output DCB must be open for this macro to execute. |
| **X'00000018'** | The interface work area (application interface block) as defined by the APWAS operand on the INTRO macro is too small to accommodate the data being copied. |
| **X'00000020'** | TCOPY did not execute because an invalid external LU name is specified in the *statname* field. Either no such entry exists in the terminal-name table or the entry is for a Group specified in the MCP. |

# TPDATE Macro

The TPDATE macro:

- Specifies whether the date, time, and origin of a message obtained by the application program are to be placed in the 16-byte area specified
- Allows the user to obtain the record delimiter from the TPROCESS entry
- Allows the user to specify whether TCAM should delete record delimiters from data going to the application program.

| Name | Operation | Operands |
|---|---|---|
| [symbol] | TPDATE | DCB=dcbname<br>,DELETE={YES}<br>{NO }<br>{,DTSAREA={area}<br>[,RECDLM= {YES}]<br>{NO } |

```
symbol
```

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

```
DCB=dcbname
```

*Function:* Specifies the symbolic name of the data control block that defines the input message queue that you wish to receive time stamped and data messages from.

*Format:* Must be the same as the symbolic label of the DCB that controls the logical input data set for which you wish messages to be time stamped and dated. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, any general register, 2 through 12, that has been loaded with the address of the DCB may be used. The register number must be enclosed within parentheses.

```
DELETE={YES}
       {NO }
```

*Function:* Specifies whether you wish the record delimiter on each incoming record to be deleted from the record before it is sent to the application program.

*Format:* YES or NO.

*Default:* DELETE = NO.

*Note:* The TPDATE macro must be coded before a GET or READ macro when you wish record delimiters to be deleted from the data coming to the application program or when the date, time, and message source information is to be provided.

```
DTSAREA=area
```

*Function:* Specifies the 16-byte area in your program where you wish TCAM to return the control information (date and time)

*Format:* Must be identical to the symbolic name specified for a work area in your program in which you wish the date and time to be entered. Register notation may be used.

*Default:* None. Specification of this operand is optional.

*Note:* *If register notation is used, any of the general registers 2 through 12 may be used. The register number must be enclosed within parentheses. You are responsible for loading the address of area into the appropriate register before the TPDATE macro executes. The format of the data that will arrive in area is:*

*YY DDD C HH MM SS NN CC CC CC CC*

*where:*
*YY = last two digits of the year (packed decimal)*
*DDD = Julian day of the year (packed decimal)*
*C = sign character (for unpacking)*
*HH = hours (packed decimal)*
*MM = minutes (packed decimal)*
*SS = seconds (packed decimal)*
*NN = hundredths of seconds (packed decimal)*
*CCCCCCCC = message source (character)*

*If the date and time are not available to TCAM, the data and time fields will contain zeros. If the source is not available, the source field will contain character blanks X'40'*

*This operand requires that DATE = YES be specified on both the TPROCESS and the PCB macros in the MCP relating to this application program.*

```
RECDLM={YES}
       {NO }
```

*Function:* Allows you to indicate whether you wish the record delimiter as specified on the TPROCESS macro in the MCP to be returned in the low-order byte of register 1.

*Format:* YES or NO.

*Default:* RECDLM = NO.

*Note:* If no record delimiter was specified on the TPROCESS macro, the value returned in the low-order byte of register 1 will be X'00'. If you specify YES, you may use the value returned in register 1 to test for this delimiter as messages are received by your program.

## Return Codes

One of the following return codes will be passed to your program in register 15 after the TPDATE macro executes:

| **Code** | **Meaning** |
|---|---|
| **X'00000000'** | TPDATE executed successfully. |
| **X'00000004'** | TPDATE did not execute because TCAM is not in the system. |
| **X'00000008'** | TPDATE did not execute because the specified DCB does not define a data set for TCAM messages. |

# WRITE Macro

The WRITE macro:

- Returns work units to the MCP after processing
- May be specified more than once in an application program.

| Name | Operation | Operands |
|---|---|---|
| `[symbol]` | `WRITE` | `,decb name`<br>`,SF`<br>`,dcb address`<br>`,area address`<br>`[,length |,'S']` |

`symbol`

*Function:* Allows symbolic addressing of this instruction within your program.

*Format:* An arbitrarily assigned character sequence of up to eight characters that must be unique within your program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this label is optional.

`areaname`

*Function:* Specifies the symbolic name of the application-program work area from which a work unit will be moved to the MCP.

*Format:* Must conform to the rules for assembler language symbols and must be identical to the symbolic name assigned to the program work area. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number must be enclosed within parentheses and the address of the work area must have been previously loaded into one of the general registers 2 through 12.

`dcbname`

*Function:* Specifies the symbolic name of the data control block defining the destination queue being written to by this macro.

*Format:* Must conform to the rules for assembler language symbols and must be identical to the symbolic name specified as the label of the data control block that defines the data set being written to. Register notation may be used.

*Default:* None. Specification of this operand is required.

*Note:* If register notation is used, the register number must be enclosed within parentheses and the address of the work area must have been previously loaded into one of the general registers 2 through 12.

`decbname`

*Function:* Specifies the symbolic name you wish to be assigned to the data event control block (DECB) created as part of this macro expansion. This symbol will be used by the CHECK macro.

*Format:* An arbitrarily assigned character sequence of up to eight characters that is unique within this program. It must also conform to the rules for assembler language symbols.

*Default:* None. Specification of this operand is required.

```
{length}
{'S'   }
```

*Function:* Allows you to specify the total length of the work unit to be transferred to the MCP. This includes the length of the work unit plus the length of any optional fields preceding the work unit.

*Format:* A decimal integer no larger than the APWAS operand specified on the INTRO macro.

*Default:* None. Specification of this operand is not required unless RECFM = U was specified on the DCB associated with this write.

*Note:* This operand is required for undefined-length (RECFM = U) work units; it is ignored for fixed-length (RECFM = F) or variable-length (RECFM = V) work units. If 'S' is specified and an undefined-length work unit is specified, the number of bytes to be written is taken from the

LRECL field of the appropriate data control block. Note that 'S' is enclosed within single quotes.

`SF`

*Function:* This operand establishes BSAM compatibility. It also specifies that the data set being written to be this instruction must have a sequential format.

*Format:* SF

*Default:* None. Specification of this operand is required.

## Return Codes

No return codes are returned in register 15 as a result of execution of this macro. Completion codes for WRITE are placed in the event control block of the DECB for this macro. (See the preceding CHECK macro discussion in this chapter.)

# Appendix A. Defining Buffers in TCAM for Application Programs

Messages being transferred between the application program work area and the MCP reside in buffers, the same as messages being transferred between the MCP and an external LU.

Although you are not responsible for defining these buffers (since the definition is made in an TCAM MCP), it helps you in coding your application program to understand how buffers are defined and what considerations are given.

## Buffer Design Considerations

The size of these application-program buffers is specified by the BUFSIZE operand of the PCB macro. This size may be overridden by the BUFL operand of the input or output DCB macro depending on which buffers you want to change.

For example, if the line buffers for all external LUs that could enter and accept messages processed by a particular application program were either 116 bytes or 232 bytes, you could define two input and two output data sets (each with its own GET/READ and PUT/WRITE process entries), one for each buffer length. You could direct all incoming messages for the application program that were entered by external LUs using 116-byte buffers to one process queue, and all incoming messages for the application program that were entered by external LUs using 232-byte buffers to the other process queue. If the system programmer coded BUFSIZE=116 in his PCB macro and BUFL=232 in the input DCB macro for the data set containing messages placed in 232-byte buffers upon arrival of the message at the MCP, no additional data transfer would be necessary when the data was read from the destination queue into the application-program buffer.

When transferring replies from the application program, you name the PUT or WRITE process entry for the 116-byte buffer output data set or for the 232-byte buffer output data set, depending upon the size of the line buffers for the reply destination. In the output DCB macro for the 232-byte buffer output data set, you would specify BUFL=232. Again, no additional data transfer would be necessary when messages were read from the destination queues into the line buffers for the destination if this scheme were followed.

## For Application Program GET or READ Buffers

The system programmer should assign the maximum number of buffers necessary (with the BUFOUT= operand of the PCB macro) to handle what you propose to be the longest messages to be transferred from MCP process queues to the application program work area at any given time. These buffers will be used to construct a *read-ahead queue,* a temporary queue in main storage on which messages are held in anticipation of a GET or READ macro from an application program. The read-ahead queue is discussed in detail in the *TCAM Installation Guide.* TCAM constructs one read-ahead queue for each process queue associated with an opened application program input data set.

The maximum capacity of any read-ahead queue is two messages. Therefore, you must assure that at least enough buffers are allocated to handle the longest message you expect to be sent. Buffers are allocated to this queue dynamically, but the queue will never contain more than the number of buffers needed to handle two messages. If the system programmer specifies a number of buffers less than that needed to contain two complete messages on the read-ahead queue, less main storage in the TCAM region is tied up by being assigned to the read-ahead queue, but more time is required to transfer messages to the application program. You must decide which alternative is more attractive to you.

The following formula for calculating the BUFOUT operand of the PCB macro will provide a read-ahead queue that is always capable of containing two complete messages; by specifying a queue of this size, delay is minimized in transferring messages to the application program. The formula is:

```
I=2X+1
```

where I represents the integer to be coded as the value for BUFOUT, and X is the maximum number of buffers needed to hold one message being transferred to the application program. The extra buffer represented by 1 is used internally by TCAM.

*Note:* If main-storage-only queuing is the only type of queuing used for process queues in the MCP, the optimum number of the buffers specified by BUFOUT is reduced; in this case, the system programmer need only specify enough buffers to handle the largest work unit expected to be sent to the application program at one time.

## For Application Program PUT or WRITE Buffers

The BUFIN operand of the PCB macro specifies the initial number of buffers that will be allocated to receive data being transferred by a PUT or WRITE macro from the application program to the MCP. (If there is more than one process entry in your program that may be referred to by PUT or WRITE macros, the number of buffers specified by BUFIN is allocated to each process entry.) Buffers assigned to receive data from an application program are allocated and sent through the incoming group of the application program message handler as they are filled by a PUT or WRITE macro.

If the number of buffers specified by BUFIN is not sufficient to handle the entire work unit being transferred, TCAM dynamically allocates additional buffers. However, such an allocation takes time; therefore, to optimize performance, a sufficient number of buffers should be assigned initially to handle the entire work unit.

Buffers are sent through the incoming group of the application-program message handler as soon as they are filled. If a buffer is not completely filled when the end of the work unit is reached, either a time or a space penalty will be incurred, depending upon whether a position field is present in the work area, and whether message or record processing is specified. (Position fields are discussed in "Defining Optional Fields in the Work Area" in Chapter 3. Message and record processing are described in "The TCAM Work Unit" in the same chapter.)

If no position field is present and message processing is specified, the partially filled buffer is sent through the incoming group of the application-program message handler as soon as the last portion of the work unit has been received. In this case, a space penalty is incurred and main storage is wasted, since the entire buffer is tied up while the work unit is being processed by the incoming group. If record processing is specified and there is no position field, a buffer that is larger than the work unit it contains is not sent through the incoming group immediately, but is held until it is filled by a subsequent PUT or WRITE macro (or until the application program signals end-of-message by closing the output data set); in this case, a time penalty is incurred.

If a position field is present and indicates that the current work unit is the last or only work unit in the message, the buffer containing that work unit is sent through the incoming group as soon as the work unit is placed in it; if the work unit is shorter than the buffer, main storage is wasted, as explained above. If the position field indicates that the current work unit is the first or an intermediate segment in a multiple segment message, then the buffer is not sent through the incoming group until it is filled or until the end of the message is encountered. If the work unit is smaller than the buffer, a time penalty is incurred, as explained previously.

Because data movement takes time, the size of the link buffers handling messages being sent to or from an application program should be the same size as the application program buffers whenever possible. By overriding the buffer size specified by the BUFSIZE operand of the PCB macro, the BUFL operand of the input and output DCB macros in the application program may tailor application program buffer sizes to match the buffer sizes for particular origin or destination external LU.

## A Coding Checklist

A good macro operand coding checklist for definition of application program buffers is:

| Macro | Operand | Description of Function |
|---|---|---|
| INTRO | UNITSZ=*integer* KEYLEN=*integer* | Specifies the length in bytes of a single buffer unit; all buffers in the TCAM system are constructed of multiples of units of this size. *integer* must be between 76 and 255 inclusive. |
| PCB | BUFSIZE=*integer* | Specifies the length in bytes of the buffers used to transfer message segments between the TCAM process queues and an application program work area. The value coded here may be overridden for a single input or output data set with the BUFL operand of the input or output DCB macro for the data set. (See the preceding discussions of the BUFL operand.) *integer* must be between 31 and 65,535 inclusive. |
| | BUFOUT=*integer* | Specifies the maximum number of application-program buffers that may be filled at one time from the destination queue. These buffers are then processed by the outgoing group of the application program message handler as a single work unit, and placed on the read-ahead queue in main storage in anticipation of a GET or READ macro from the application program. *integer* must be at least 2 (TCAM uses one buffer internally) and may be no greater than 15. |
| | BUFIN=*integer* | Specifies the initial number of buffers to be allocated to a message handler to receive data being transferred by a PUT or WRITE macro from the application program work area.<br><br>Contrast this discussion with the BUFOUT discussion above. *integer* must be between 2 and 15 inclusive. |
| | RESERVE= (*integer1, integer2*) | Specifies the number of bytes to be reserved in the beginning of each buffer. This space is for the insertion of time and date for each message. *integer1* and *integer2* can each be any decimal value between 1 and 255. Data will be inserted in these fields by the DATETIME or SEQUENCE macros in the MCP. |
| Application Program Input to DCB | BUFL=*integer* | Specifies the length (in bytes) of the buffers used to transfer message segments from the MCP to the application program where this DCB resides; this value overrides the value specified by the BUFSIZE operand of the PCB macro. *integer* must be between 35 and 65,535 inclusive. |
| Application Program Output DCB | BUFL=*integer* | Specifies the length (in bytes) of the buffers to be used to transfer message segments from the application program where this DCB resides to the MCP; this value overrides the value specified by the BUFSIZE operand of the PCB macro. *integer* must be between 35 and 65,535 inclusive. |

# Appendix B. Checklist of Possible Coding Errors

The following list of questions are intended to aid in the diagnosis of possible errors in coding an application program. Five reference headings are provided to ease the location of a possible error: Activation and Deactivation of Input and Output Data Sets and the MCP, Work Areas, Message Transfers, Message Queues, and Buffers.

The list of possible errors in coding an application program is in the form of questions with YES/NO answers so that you can examine your code against the correct procedures.

| **Question** | **Right** | **Wrong** |
|---|---|---|
| I. ACTIVATION AND DEACTIVATION OF INPUT AND OUTPUT DATA SETS AND THE MCP | | |
| 1. Did you follow standard linkage conventions? | YES | NO |
| 2. If the application program is linked as AUTHORIZED, then code AUTHA=YES on the INTRO macro. | YES | NO |
| 3. Did you code an OPEN macro for each DCB? | YES | NO |
| 4. Did you check each OPEN macro for successful completion? | YES | NO |
| 5. Did you issue an OPEN macro for a PCB? | NO | YES |
| 6. Did you code closedown procedures? | YES | NO |
| 7. Do your application programs have lower priority than your MCP? | YES | NO |
| 8. Did you specify a record delimiter for variable-length records or messages? | YES | NO |
| 9. Did you activate your application program before you started your MCP? | NO | YES |
| 10. Did you omit any DD statements? | NO | YES |
| 11. Did you check all return codes provided by TCAM? | YES | NO |

| Question | | | Right | Wrong |
|---|---|---|---|---|
| | 12. | Have all GET/READ and PUT/WRITE DCBs been opened in this task or the attaching (mother) task? | YES | NO |
| II. | WORK AREAS | | | |
| | 1. | Did you destroy or overlay your work-area prefix? | NO | YES |
| | 2. | Is your work-area size compatible with TCAM buffer size? | YES | NO |
| | 3. | Is the value specified by the APWAS operand equal to or larger than the largest work-area size of any application program macro. | YES | NO |
| | 4. | Is your work-area size for copy functions large enough when using the TCOPY macro or when displaying the option fields by an operator control command? | YES | NO |
| | 5. | Did you omit the BLKSIZE operand of the DCB macro for GET in locate mode? | NO | YES |
| | 6. | If you specified DATE=YES with your TPROCESS macro, did you also specify DATE=YES with the PCB macro and the DTSAREA operand of the TPDATE macro? | YES | NO |
| | 7. | Did you code a TPROCESS macro with the QBACK operand coded for each application program that will issue a QRESET macro? | YES | NO |
| | 8. | If you are using multiple retrieve, did you either (1) specify a work-area large enough or (2) issue subsequent READs or GETs to handle the complete message? | YES | NO |
| III. | MESSAGE TRANSFERS | | | |
| | 1. | Are your incoming and outgoing work units compatible? | YES | NO |
| | 2. | Is your destination correct for a lock response? | YES | NO |
| | 3. | Did you code an outmessage subgroup in your message handler (MH)? | NO | YES |

| Question | | | Right | Wrong |
|---|---|---|---|---|
| | 4. | Did you specify an external LU to receive response messages from basic operator commands generated by the application program (in the ALTDEST operand of the TPROCESS macro or in the RESPDEST operand of the CODE or IEDOPCTL macro)? | YES | NO |
| | 5. | Did you specify a work-unit size for PUT or WRITE? | YES | NO |
| | 6. | When you are using message processing, did you specify the OPTCD=U operand of the DCB macros? | YES | NO |
| | 7. | If you specified OPTCD=W with the input DCB macro, did you make your work unit 8 bytes larger than the buffer size defined by the DCB macro? | YES | NO |
| | 8. | Was the same PCB specified for the GET and PUT response TPROCESS entries that were used for lock mode? | YES | NO |
| IV. | MESSAGE QUEUES | | | |
| | 1. | Did you code the QUEUES operand of the TPROCESS macro for GET or READ? | YES | NO |
| | 2. | Did you code the QUEUES operand of the TPROCESS macro for PUT or WRITE? | NO | YES |
| | 3. | If the TPDATE macro is used, did you code it after OPEN but before GET or READ? | YES | NO |
| | 4. | If you are using the MCOUNT macro, did you code it after OPEN? | YES | NO |
| V. | BUFFERS | | | |
| | 1. | Did you specify enough buffer units? | YES | NO |

# Glossary

This glossary includes terms and definitions from the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems,* GC20-1699. Definitions from the *American National Dictionary for Information Processing* are identified by an asterisk (*).

## Reference Words Used in the Entries

The following reference words are used in this glossary.

*Contrast with.* Refers to a term that has an opposed or substantively different meaning.

*Deprecated term for.* Indicates that the term should not be used. It refers to a preferred term, which is defined.

*See.* Refers to multiple-word terms that have the same last word.

*See also.* Refers to related terms that have similar (but not synonymous) meanings.

*Synonym for.* Appears in the commentary of a less desirable or less specific term and identifies the preferred term that has the same meaning.

*Synonymous with.* Appears in the commentary of a preferred term and identifies less desirable or less specific terms that have the same meaning.

**ACB.** In VTAM, access method control block.

**accept.** In a VTAM application program, to accept a CINIT request from a system services control point (SSCP) to establish a session with a logical unit; the application program acts as the primary end of the session. Contrast with *acquire (1).*

*Note:* The accept process causes a BIND request to be sent from the primary end of the session to the logical unit that will act as the secondary end of the session, requesting that the session be established and passing session parameters. For example, the session-initiation request that originally caused the SSCP to send the CINIT request may have resulted from a logon by the terminal operator, from a macro instruction issued by a VTAM application program, or from a VTAM operator command.

**access method.** A technique for moving data between main storage and input/output devices. See *Basic Direct Access Method, Basic Sequential Access Method, TCAM (Version 2 and previous releases), and VTAM.*

**access method control block (ACB).** A control block that links an application program to VSAM or VTAM.

**access method interface (AMI).** The TCAM function for managing communication on the access method control block (ACB) interface between TCAM and VTAM.

**ACF.** Advanced Communications Function.

**ACF/TCAM.** Advanced Communications Function for TCAM. Synonym for *TCAM.*

**ACF/VTAM.** Advanced Communications Function for VTAM. Synonym for *VTAM.*

**acquire.** (1) In VTAM, the operation in which an authorized VTAM application program initiates and establishes a session with another logical unit; the application program acts as the primary end of the session. Contrast with *accept.* (2) In relation to VTAM resource control, to take over resources (communication controllers or other physical units) that were formerly controlled by a data communication access method in another domain, or to assume control of resources that were controlled by this domain but released. Contrast with *release.* See also *resource takeover.*

**active.** In VTAM, the status of a resource that makes the resource known to VTAM (for major nodes) or makes it available for use in the network (for minor nodes). For a logical unit (LU) minor node, it also enables the LU to participate in LU-LU sessions. Contrast with *inactive.*

**address space.** The area of virtual storage that is available for a particular job.

**Advanced Communications Function (ACF).** A group of IBM program products, principally VTAM, TCAM, NCP, and SSP.

**Advanced Communications Function for the Network Control Program (NCP).** An IBM program product that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. Programs (SSP)

**Advanced Communications Function for Systems Support Programs (SSP).** An IBM program product made up of a collection of utilities and small programs. SSP is required for the operation of the NCP.

**Advanced Communications Function for TCAM (TCAM).** An IBM program product that provides queued message handling. TCAM, Versions 1 and 2, are telecommunications access methods, but TCAM, Version 3, is a message handling subsystem.

**Advanced Communications Function for VTAM (VTAM).** An IBM program product that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability. VTAM runs under MVS (OS/VS1 and OS/VS2), VSE, and VM/SP. It supports direct control application programs and subsystems such as VSE/POWER.

**affinity-based routing.** Message routing in which a temporary relationship, or routing affinity, is established between a source and a destination; all messages from the source are routed to the destination for the duration of the relationship. See also *invariant routing, transaction-based routing, routing by destination, routing by key.*

**AMH.** Application message handler.

**AMI.** Access method interface.

**application message handler (AMH).** A user-defined routine that processes messages that are received by the message control program (MCP) from an application program or that are sent by the MCP to an application program. See *message handler, device message handler, internodal message handler.* See also *Message control program.*

**application program.** (1) A program written for or by a user that applies to the user's work. (2) A program used to connect and communicate with resources in a network, enabling users to perform application-oriented activities.

**asynchronous.** Without regular time relationship; unexpected or unpredictable with respect to the execution of a program's instructions.

**automatic purge/copy/redirect.** A collection of message-handler and extended operator control functions that permits messages to be conditionally or unconditionally redirected to another destination, copied to another destination, or purged (that is, not sent to any destination).

**Basic Direct Access Method (BDAM).** An access method used to directly retrieve or update particular blocks of a data set on a direct access device.

**basic information unit (BIU).** In SNA, the unit of data and control information that is passed between half-sessions. It consists of a request/response header (RH) followed by a request/response unit (RU).

**basic operator command.** An operator command directed to the basic operator control system service program. Synonymous with *basic operator control command.*

**basic operator control.** The function of a particular system service program that processes a set of basic operator commands. These commands allow the operator to determine the status of the TCAM system and to alter, start, and stop TCAM and its resources by entering appropriate commands from either the system console or a basic operator control station. The basic operator control system service program is required in order to execute a TCAM message control program (MCP).

**basic operator control command.** Synonym for *basic operator command.*

**basic operator control station.** A system console, external logical unit (LU), or application program that is authorized to enter operator commands to be executed by the basic operator control system service program.

**basic primary operator control station.** A basic operator control station that is sent all TCAM error-recovery messages and TCAM reply messages to basic operator commands. See *basic secondary operator control station, extended primary operator control station, extended secondary operator control station.*

**basic secondary operator control station.** A basic operator control station that is sent only the reply messages to basic operator commands entered from it. See *basic primary operator control station, extended primary operator control station, extended secondary operator control station.*

**Basic Sequential Access Method (BSAM).** An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential access or direct access device.

**BDAM.** Basic Direct Access Method.

**begin bracket.** In SNA, the value (binary 1) of the begin-bracket indicator in the request header (RH) of the first request in the first chain of a bracket; the value denotes the start of a bracket. Contrast with *end bracket.* See also *bracket.*

**bidder.** In SNA, the LU-LU half-session defined at session activation as having to request and receive permission from the other LU-LU half-session to begin a bracket. Contrast with *first speaker.* See also *bracket protocol.*

**binary synchronous communication (BSC).** Communication using binary synchronous line discipline. See also *binary synchronous transmission.*

**binary synchronous transmission.** Data transmission in which synchronization of characters is controlled by time signals generated at the sending and receiving stations. Contrast with *start-stop transmission, synchronous data link control.*

**bind image.** Synonym for *logon mode.*

**bind image table.** Synonym for *logon mode table.*

**BIU.** Basic information unit.

**BIU segment.** In SNA, a portion of a basic information unit (BIU) that is contained within a path information unit. It consists of either a request/response header (RH) followed by all or a portion of a request/response unit (RU), or of only a portion of an RU. See also *segment.*

**bracket.** In SNA, one or more chains of request units (RUs) and their responses that are exchanged between two LU-LU half-sessions and that represent a transaction between them. A bracket must be completed before another bracket can be started. Examples of brackets are data base inquiries/replies, update transactions, and remote job entry output sequences to work stations. See *begin bracket, end bracket.* See also *RU chain.*

**bracket protocol.** In SNA, a data flow control protocol in which exchanges between two LU-LU half-sessions are achieved through the use of brackets, with one logical unit (LU) designated at session initiation as the first speaker and the other LU as the bidder. The bracket protocol involves bracket initiation and termination rules. See also *bidder, first speaker.*

**bracket state manager.** A TCAM routine that enforces the bracket protocol by making proper bracket state changes and detecting bracket errors.

**broadcast.** The simultaneous transmission of data to a number of destinations.

**BSAM.** Basic Sequential Access Method.

**BSC.** Binary synchronous communication.

**buffer.** (1) * A routine or storage area used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another. (2) An area of storage that is temporarily reserved for use in performing an input/output operation, into which data is read or from which data is written.

**buffer list.** In VTAM, a contiguous set of control blocks (buffer list entries) that allow an application program to send function management data (FMD) from a number of discontiguous buffers with a single SEND macro instruction.

**buffer prefix.** A control area within each buffer that contains buffer control information. A user must allow room for the buffer prefix when specifying buffer size.

**buffer unit.** The smallest block of main storage from which TCAM buffers and main storage message queues can be built. Synonymous with *main storage unit.*

**buffer-unit pool.** All the buffer units in a particular TCAM system.

**cascade entry.** A terminal-table entry associated with a cascade list.

**cascade list.** A list of pointers to single entries. When a cascade entry is named as the destination for a message, the message is sent to the valid entry in the list with the fewest messages queued for it.

**chain.** See *RU chain.*

**channel program block (CPB).** A TCAM control block used in the transfer of data between buffer units and message queues maintained on disk.

**checkpoint data set.** An optional TCAM data set that contains the checkpoint records used to reconstruct the message control program (MCP) environment after closedown or system failure when the TCAM checkpoint/restart service facility is used.

**checkpoint records.** Records that contain the status of a job and the system at the time the records are written by the checkpoint routine. These records provide the information necessary for restarting a job without having to return to the beginning of the job. There are four types: checkpoint request record, control record, environment record, and incident record.

**checkpoint request record.** A checkpoint record taken as a result of the execution of a CKREQ macro instruction in an application program; the record contains the status of a single destination queue for the application program. See *control record, environment record, incident record.*

**checkpoint/restart service facility.** A TCAM service facility that records the status of the TCAM system at designated intervals or following certain events. After system failure, the TCAM system can be restarted and can continue without loss of messages.

**CIB.** Command input buffer.

**clear data.** Data that is not enciphered.

**clear session.** A session in which only clear data is transmitted or received. Contrast with *cryptographic session.*

**closed subroutine.** A subroutine of which one replica suffices for the subroutine to be linked by calling sequences for use at more than one place in a computer program. Contrast with *open routine.*

**closedown.** The orderly termination of the message control program. See *flush closedown, quick closedown.*

**cold restart.** Startup of a message control program (MCP) following either a flush closedown, a quick closedown, or a system failure. A cold restart ignores the previous environment; that is, the MCP is started as if this were the initial startup. A cold restart is the only type of restart possible when no checkpoint/restart service facility is used. Contrast with *warm restart.*

**command.** (1) In SNA, any field set in the transmission header, request header (RH), and sometimes portions of a request unit (RU), that initiates an action or begins a protocol; for example, (a) Bind Session (session-control request unit), a command that activates an LU-LU session; (b) the change-direction indicator in the RH of the last RU of a chain; (c) the virtual route reset window indicator in an FID4 transmission header. (2) Loosely, a request unit. (3) In SDLC, the control information (in the C-field of the link header) sent from the primary station to the secondary station. (4) In TCAM, an operator control command.

**command input buffer (CIB).** A buffer-like area that contains operator commands entered at the system console. Main storage space is allocated for it dynamically and is freed once the operator command contained within the CIB has been processed. Only one CIB need be specified for operator commands entered from the system console.

**communication common carrier.** In the USA and Canada, a public data transmission service that provides the general public with transmission service facilities; for example, a telephone or telegraph company.

**COMWRITE.** An IBM-supplied subtask of the TCAM initiator that formats and writes trace records to the COMWRITE data set.

**COMWRITE data set.** A data set on a sequential storage device in which trace information is written.

**connection point manager.** In SNA, a component of the transmission control layer that (a) performs session-level pacing of normal-flow requests; (b) checks sequence numbers of received response units; (c) verifies that request units do not exceed maximum permissible size; (d) routes incoming request units to their destinations within the half-session; and (e) enciphers and deciphers function management data (FMD) request units when cryptography is selected. The sending connection point manager within a half-session builds the request/response header for outgoing request units, and the receiving connection point manager interprets the request headers that precede incoming request units.

**control record.** A checkpoint record included in a checkpoint data set that keeps track of the correct environment records, incident records, and checkpoint request records to use for restructuring the message control program environment during restart. See *environment record, incident record, checkpoint request record.*

**conversational mode.** A mode in which the next message received by an external logical unit (LU) after it enters an inquiry message is a reply to that message. See *lock mode.*

**CPB.** Channel program block.

**cryptographic.** Pertaining to the transformation of data to conceal its meaning.

**cryptographic session.** An LU-LU session in which a function management data request may be enciphered before it is transmitted, and deciphered after it is received. Contrast with *clear session.* See also *mandatory cryptographic session, selective cryptographic session.*

**DASD.** Direct access storage device.

**data control block (DCB).** A control block used by access method routines in storing and retrieving data.

**Data Encryption Standard (DES) algorithm.** A cryptographic algorithm designed to encipher and decipher 8-byte blocks of data using a 64-bit cryptographic key, as specified in the *Federal Information Processing Standard Publication 46,* January 15, 1977.

**data flow.** (1) In SNA: any of four flows in a given session: primary-to-secondary flow, secondary-to-primary flow, normal flow, or expedited flow. (2) The type of route or extended route that a message takes from its origin to its destination, including the host nodes that process the message while it is enroute to its destination. See *level 1 data flow, level 2 data flow, level 2+ data flow, level 3 data flow.*

**data flow control (DFC) layer.** In SNA, the layer within a half-session that (a) controls whether the half-session can send, receive, or concurrently send and receive request/response units (RUs); (b) groups related RUs into RU chains; (c) delimits transactions via the bracket protocol; (d) controls the interlocking of requests and responses in accordance with control modes specified at session activation; (e) generates sequence numbers; and (f) correlates requests and responses.

**data set.** (1) The major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. (2) * Deprecated term for *modem.*

**data staging.** An extended networking technique in which high-volume, low-priority message traffic is moved from one TCAM node to another, progressively approaching the destination TCAM. Data staging allows such traffic to be moved at a convenient time to avoid overloading the network to protect response times for high-priority inquiries.

**DCB.** Data control block.

**dead-letter queue.** In TCAM, a queue containing messages that could not be placed in the appropriate destination queue.

**decipher.** To convert enciphered data into clear data. Contrast with *encipher.* Synonymous with *decrypt.*

**decrypt.** To convert encrypted data into clear data. Contrast with *encrypt.* Synonym for *decipher.*

**definite response.** In SNA, a form of response requested in the request header (RH) for a request unit (RU). The receiver is requested to return a response indicating whether the request is acceptable as received. Contrast with *exception response, no response.*

**delayed-request mode.** In SNA, an operational mode in which the sender may continue sending request units on the normal flow after sending a definite-response chain on that flow, without waiting to receive the response to that chain. Contrast with *immediate-request mode.* See also *delayed-response mode.*

**delayed-response mode.** In SNA, an operational mode in which the receiver of normal-flow request units can return responses to the sender in a sequence different from that in which the corresponding request units are sent. Contrast with *immediate-response mode.* See also *delayed-request mode.*

**delimiter macro instruction.** A TCAM message-handler macro instruction that classifies and identifies sequences of functional message-handler macro instructions and directs control to the appropriate sequence of functional macro instructions. Contrast with *functional macro instruction.*

**DES.** Data Encryption Standard. See also *Data Encryption Standard algorithm.*

**DES algorithm.** Data Encryption Standard algorithm.

**destination.** An external logical unit (LU) or application program to which a message or other data is directed.

**destination queue.** A queue on which messages bound for a particular destination are placed after being processed by the incoming group of the message handler. See also *process queue.*

**device message handler (DMH).** A user-written routine defined in a TCAM message control program (MCP) that processes messages being received from or sent to an external logical unit (LU). See also *application message handler, internodal message handler, message handler.*

**DFC.** Data flow control.

**direct access storage device (DASD).** A storage device in which the access time is effectively independent of the location of the data.

**distribution entry.** A terminal-table entry associated with a distribution list.

**distribution list.** A list of pointers to single or cascade entries. When a distribution entry is named as the destination for a message, the message is sent as separate transmissions to all items in the list.

**DMH.** Device message handler.

**domain.** In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all the associated resources that the SSCP has the ability to control by means of activation requests and deactivation requests. Svnonymous with *single-domain network.*

**dynamic accounting facility.** A TCAM service facility that gathers resource utilization data for processing by user-supplied applications.

**EBCDIC.** Extended binary-coded decimal interchange code.

**encipher.** (1) To scramble data or convert it, prior to transmission, to a secret code that masks the meaning of the data to any unauthorized recipient. (2) In VTAM, to convert clear data into enciphered data. Contrast with *decipher.* Synonymous with *encrypt.*

**enciphered data.** Data that is intended to be illegible to all except those who legitimately possess the means to reproduce the clear data.

**encrypt.** In VTAM, to convert clear data into enciphered data. Contrast with *decrypt.* Synonym for *encipher.*

**end bracket.** In SNA, the value (binary 1) of the end bracket indicator in the request header (RH) of the first request of the last chain of a bracket. The value denotes the end of the bracket. Contrast with *begin bracket.* See also *bracket.*

**end-of-address (EOA) character.** A character that must be placed in a message if the system is to route that message to several destinations. The character must immediately follow the last destination coded in the message header.

**end-to-end session.** In TCAM a logical connection in which an affinity-based routing relationship has been established between a source and a destination. Either the source or destination can be either a logical unit (LU) or an application program. End-to-end sessions require routing by key.

**end user.** The ultimate source or destination of application data flowing through an SNA network. An end user may be an application program or a terminal operator.

**environment record.** A checkpoint record of the total TCAM system at a single point in time. See *checkpoint request record, checkpoint/restart service facility, control record, incident record.*

**EOA.** End-of-address. See *end-of-address character.*

**error record.** Five bytes assigned to each message processed by a message handler. These bytes indicate physical or logical errors that have occurred during transmission or during subsequent processing or queuing of the message. In addition, a message error record may be the created when a session cannot be established. Error records are checked by error-handling macro instructions in the in-message and out-message subgroups of a message handler. Synonymous with *message error record.*

**error-recovery procedures.** A set of internal TCAM routines that attempt to recover from transmission errors.

**exception request.** In SNA, a request that replaces another message unit in which an error has been detected.

**exception response.** In SNA, a value in the form-of-response requested field of a request header. The receiver is requested to return a response only if the request is unacceptable as received or cannot be processed. That is, a negative response, but not a positive response, may be returned. Contrast with *definite response, no response.* See *negative response.*

**expedited flow.** In SNA, a data flow designated in the transmission header that is used to carry network control, session control, and various data flow control request/response units (RUs). The expedited flow is separate from the normal flow (which carries primarily end-user data) and can be used for commands that affect the normal flow on the path. Contrast with *normal flow*. See also *isolated pacing response*.

**extended lock mode.** A type of lock mode in which an external logical unit (LU) remains in lock mode for the duration of several inquiry/reply cycles. Contrast with *message lock mode*. See *lock mode*.

**extended network.** A network that includes two or more TCAM systems using extended networking facilities.

**extended networking.** A TCAM function that uses a collection of TCAM macro instructions, system service programs, and message-handler facilities to simplify TCAM system definition, management, and error recovery in a network with two or more TCAM systems.

**extended operator command.** An operator command directed to the extended operator control system service program. Synonymous with *extended operator control command*.

**extended operator control.** The function of a particular system service program that processes a set of extended operator commands. These commands are not required in order to control a TCAM system, but are useful in some environments. The extended operator control system service program is required if the message control program (MCP) uses one or more of the following functions: (a) extended networking, (b) online retrieval system service program, or (c) automatic purge/copy/redirect.

**extended operator control command.** Synonym for *extended operator command*.

**extended operator control station.** A system console, external logical unit (LU), or application program that is authorized to enter extended operator commands. See *basic operator control station, extended primary operator control station, extended secondary operator control station*.

**extended primary operator control station.** An extended operator control station that receives the extended operator control startup and closedown messages; responses to extended operator commands entered from it; responses to extended operator commands that successfully modify the TCAM system; and, optionally, the online retrieval system service program startup and closedown messages (if online retrieval is part of the TCAM system). See *basic primary operator control station, basic secondary operator control station, extended secondary operator control station*.

**extended secondary operator control station.** An extended operator control station that enters extended operator commands and receives the responses made to those commands. See *basic primary operator control station, basic secondary operator control station, extended primary operator control station*.

**extended route.** In TCAM extended networking, a series of one or more routes that involves an intermediate TCAM node.

**external LU.** A logical unit (LU) that communicates with a TCAM message control program (MCP) through VTAM. Each external LU is defined to the MCP with a TERMINAL macro instruction.

**FHP.** Fixed header prefix.

**first speaker.** In SNA, the LU-LU half-session defined at session activation as (a) able to begin a bracket without requesting permission from the other LU-LU half-session to do so, and (b) winning contention if both half-sessions attempt to begin a bracket simultaneously. Contrast with *bidder*. See also *bracket protocol*.

**fixed header prefix (FHP).** An optional control block that provides a place to keep message-related information needed by certain optional TCAM functions.

**flush closedown.** A closedown of TCAM during which incoming message traffic is suspended and queued outgoing messages are sent to their destinations ("flushed" from the message queues) before closedown is completed. Contrast with *quick closedown*.

**FM.** Function management.

**FMD.** Function management data.

**FMD services layer.** In SNA, the layer within a half-session that routes function management data (FMD) requests and responses to particular network addressable unit (NAU) services manager components and that provides session network services or session presentation services, depending on the type of session.

**function management data (FMD) services.** A general term for session network services and session presentation services, both of which process FMD requests and responses.

**function management (FM) header.** In SNA, one or more headers, optionally present in the leading request units (RU) of an RU chain, that allow one half-session in an LU-LU session to: (a) select a destination (for example, a program or a device) as the session partner and control the way that the end-user data it sends is handled at the destination, (b) change the destination or the characteristics of the data during the session, and (c) transmit between session partners status or user information about the destination (for example, a program or a device).

**function management (FM) profile.** In SNA, a specification of various data flow control protocols (such as request unit chains and data flow control requests) and function management data (FMD) options (such as use of FM headers, compression, and alternate codes) supported for a particular session. Each function management profile is identified by a number.

**functional macro instruction.** A TCAM macro instruction that performs the specific operations required for messages directed to the message handler. Contrast with *delimiter macro instruction.*

**group entry.** A terminal-table entry associated with a group of logical units (LUs).

**group of logical units (LUs).** In TCAM, a set of external LU definitions that are associated with the same group entry. See also *group entry.*

**half-session.** In SNA, a component that provides FMD services, data flow control, and transmission control for one of the sessions of a network addressable unit. See *primary half-session, secondary half-session.*

**header.** That portion of a message containing control information for the message; a header might contain one or more destination fields, the name of the originating station, an input sequence number, a character string indicating the type of message, and a priority level for the message. The message header is operated on by macro instructions in the inheader and outheader subgroups of the message handler. See *message header.*

**header buffer.** A buffer containing either all or the first part of a message header. Contrast with *text buffer.*

**host computer.** Synonym for *host processor.*

**host logical unit (LU).** An SNA logical unit (LU) located in a host processor, for example, a VTAM application program. See *TCAM host logical unit (LU).*

**host node.** In SNA, a subarea node that contains a system services control point (SSCP).

**host processor.** The controlling processor with its operating system, access methods, and application programs. A system services control point is located in a host processor. Synonymous with *host computer.*

**IMH.** Internodal message handler.

**immediate-request mode.** An operational mode in which the sender, after sending a definite-response request chain on a given flow, stops sending request units on the flow until the chain has been responded to. Contrast with *delayed-request mode.* See *immediate-response mode.*

**immediate-response mode.** An operational mode in which the receiver responds to request units on a given normal flow in the order it receives them; that is, in a first-in, first-out sequence. Contrast with *delayed-response mode.* See *immediate-request mode.*

**inactive.** In VTAM, pertaining to a resource that has never been activated or has been deactivated by a VTAM operator command. Contrast with *active.*

**inblock subgroup.** The part of a message handler (MH) incoming group that, if used, precedes the inheader subgroup and blocks several physical messages into a longer, logical message or unblocks a physical message into a shorter, logical message.

**inbuffer subgroup.** The part of a message handler (MH) incoming group that operates on each segment of an incoming message.

**incident record.** A checkpoint record that logs a change in external logical unit (LU) or application program status, and in the contents of an option field that occurred since the last environment record was taken. It is used to update the information contained in environment records at restart after a closedown or system failure. See *checkpoint request record, control record, environment record.*

**incoming group.** That portion of a message handler that is designed to handle incoming messages for the message control program (MCP). Contrast with *outgoing group.*

**incoming message.** A message sent from an external logical unit (LU) or application program to the message control program (MCP).

**inheader subgroup.** The part of a message handler (MH) incoming group that operates on all or part of an incoming message header.

**initial chaining value (ICV).** An eight-byte, pseudo-random number used to verify that both ends of a session with cryptography have the same session cryptography key. The initial chaining value is also used as input to the Data Encryption Standard (DES) algorithm to encipher or decipher data in a session with cryptography.

**initiation.** Synonym for *LU-LU session initiation.* See also *session-initiation request.*

**initiator.** The component of TCAM that is executed as the job-step task. The initiator starts, monitors, and restarts the message control program (MCP), TCAM system service programs, and user-supplied system service programs. It can also display status information at the system console.

**inmessage subgroup.** The part of a message handler (MH) incoming group that specifies actions to be taken after a complete message has arrived at the message control program (MCP).

**input data set.** A data set that contains all messages or records sent to an application program from a single process queue. Contrast with *output data set.*

**inquiry/reply.** A TCAM application in which a device message handler receives a message from an external logical unit (LU) and then routes it to an application program that processes the message and generates a reply. The reply is routed back to the inquiring external LU.

**intercepted resource.** An external logical unit (LU) to which no messages may be sent for a specified time interval or until an operator command or an application-program macro instruction is issued to release messages. An intercepted resource can enter messages, but messages destined for it are not sent.

**intermediate function.** In SNA, a path control capability within a subarea node that receives and routes path information units that neither originate in nor are destined for the network addressable units in that subarea node.

**intermediate TCAM node.** In TCAM extended networking, a TCAM node that processes messages flowing along the extended route but does not provide the queuing for the originating or destination resources for those messages. Processing by an intermediate TCAM node includes processing by the incoming group of the internodal message handler, queuing of each message on the internodal destination queue for the next TCAM node on its extended route, and processing by the outgoing group of the internodal message handler (IMH). See *TCAM node.*

**internodal awareness.** In TCAM extended networking, a function used by TCAM systems to share information about each other. This information includes the status of TCAM systems, the status of application programs in TCAM systems, and the contents of selected key-table entries. This function is provided by node path system service programs in the various TCAM systems that communicate with each other.

**internodal destination queue.** In TCAM extended networking, a destination queue for an external logical unit (LU) that is a partner in a utility session.

**internodal message handler (IMH).** In TCAM extended networking, a message handler that processes messages flowing on utility sessions.

**internodal sequence number synchronization.** In TCAM extended networking, the function of a particular system service program that operates in conjunction with the internodal message handler. Internodal sequence number synchronization is used to request retransmission from any TCAM node of sequence-numbered messages not received on that utility session and retransmit sequence-numbered messages flowing on a utility session when requested to do so by another TCAM node or an extended operator command.

**internodal sequence prefix.** In TCAM extended networking, a control block that is used to contain sequence-number information for messages flowing on utility sessions.

**invariant routing.** Message routing in which messages from the same source are always sent to the same destination. See *affinity-based routing, transaction-based routing.* See also *routing by destination, routing by key.*

**key.** A character string that matches a definition in the key table. This key identifies the destination of a message or special processing to be done on that message. See also *key table.*

**key table.** A main-storage table of keys and their definitions, which contain information on routing and special processing to be done on a message. See also *key*.

**layer.** In SNA, a grouping of related functions that are logically separate from the functions in other layers; the implementation of the functions in one layer can be changed without affecting other layers. See *data flow control layer, FMD services layer*.

**LCB.** Line control block.

**level 1 data flow.** In SNA, a data flow (within a single-domain network) in which each message's origin and destination logical units (LUs) reside in the same domain.

**level 2 data flow.** In TCAM extended networking, a data flow on an extended route in which each message enters the TCAM node that provides queuing for the originating resource, another TCAM node that provides queuing for the destination resource, and one or more intermediate TCAM nodes.

**level 2+ data flow.** In TCAM extended networking, a data flow on an extended route in which each message enters both the TCAM node that provides queuing for the originating resource and the TCAM node that provides queuing for the destination resource, but does not pass through any intermediate TCAM nodes.

**level 3 data flow.** In SNA, a data flow (within a multiple-domain network) in which each message's origin and destination logical units (LUs) reside in different domains.

**line.** Any physical medium, such as a wire or telephone circuit, that connects one or more stations to a communication control unit or connects one communications control unit with another. See also *link*.

**line control block (LCB).** A control block used for scheduling, sending, and receiving.

**link.** In SNA, the combination of the link connection and the link stations joining network nodes; a serial-by-bit connection under the control of SDLC. A link connection is a physical medium of transmission, such as a telephone wire or a microwave beam. A link includes a physical medium of transmission (a line), a protocol (SDLC), and associated communication devices and programming; it is both logical and physical.

**load balancing.** In TCAM extended networking, the technique for balancing the message flow between any pair of TCAM nodes by assigning different paths to different messages flowing between them.

**lock mode.** A mode in which an external logical unit (LU) entering an inquiry message for an application program is ensured that the next message it receives is a reply from the application program. See *conversational mode, extended lock mode, message lock mode*.

**log.** A collection of messages or message segments placed on a secondary-storage device for accounting or data collection purposes.

**log data set.** A data set consisting of the messages or message segments recorded on a secondary-storage medium by the TCAM logging facility.

**logging service facility.** A TCAM service facility that selectively causes incoming or outgoing messages or message segments to be copied onto tape or disk. The log produced by the logging service facility provides a record of message traffic through the message control program (MCP).

**logical message.** A user-defined message, consisting of one or more related units of data in a transmission, ending with an end-of-message code. Contrast with *physical message*.

**logical unit (LU).** In SNA, a port through which an end user gains access to the SNA network to communicate with another end user and through which the end user uses the functions provided by the SSCP. An LU can have at least two sessions--one with the SSCP, and one with another LU--and may be able to have many sessions with other LUs. Contrast with *physical unit*. See *host LU, auxiliary LU, primary logical unit, secondary logical unit*. See also *system services control point*.

**logical unit (LU) services.** In SNA, capabilities in a logical unit to: (a) receive requests from an end user and, in turn, issue requests to perform the requested functions, typically for session initiation; (b) receive requests from the SSCP to activate LU-LU sessions through Bind Session requests; and (c) provide session presentation and other services for LU-LU sessions.

**logon mode.** In VTAM, a set of session parameters specified in a logon mode table entry for communication with a logical unit. See also *session parameters*. Synonymous with *bind image*.

**logon mode table.** In VTAM, a set of entries for one or more logon modes. Each logon mode is identified by a logon mode name. Synonymous with *bind image table*.

**logtype entry.** A terminal-table entry associated with a queue on which complete messages reside while awaiting transfer by the logging service facility. A logtype entry is not needed if message segments only are to be logged.

**LU.** Logical unit.

**LU-LU half-session.** A half-session in which the session involved is an LU-LU session.

**LU-LU session.** In SNA, a session between two logical units (LUs) in an SNA network. It provides communication between two end users, or between an end user and an LU services component.

**LU-LU session initiation.** The process that begins with a session-initiation request from a logical unit (LU) to a system services control point and culminates in activation of an LU-LU session. See also *session activation.*

**LU-LU session termination.** The process that begins with either a session-termination request from a logical unit to a system services control point, or an Unbind request from one logical unit to another, and that culminates in deactivation of an LU-LU session. See also *session deactivation.*

**LU-LU session type.** The classification of an LU-LU session in terms of the specific subset of SNA protocols and options required by the logical units for that session. LU-LU session types 0, 1, 2, 3, 4, 6, and 7 are defined in SNA.

**LU services.** See *logical unit services.*

**LU services manager.** An SNA component that provides a logical unit (LU) with network services and end-user to end-user services. The LU services manager provides services for all half-sessions within the LU.

**main storage unit.** Synonym for *buffer unit.*

**mandatory cryptographic session.** A cryptographic session in which all outgoing data is enciphered and all incoming data is deciphered. Contrast with *selective cryptographic session.*

**MCP.** Message control program.

**MCP definition.** The collection of macro-language statements by which the network is defined to TCAM in the resource-definition section of the message control program (MCP).

**message.** In TCAM, a unit of data transmitted from one point to another. See *logical message, physical message.*

**message control program (MCP).** A general term referring to any specific implementation of TCAM, including initialization and termination routines, resource management routines, message handling routines, and service facilities.

**message error record.** Synonym for *error record.*

**message handler (MH).** A sequence of user-specified macro instructions and basic assembler language instructions that invoke routines that examine and process control information in message headers and perform functions necessary to prepare messages for forwarding to their destinations. See *application message handler, device message handler, internodal message handler.* See also *delimiter macro instruction, functional macro instruction.*

**message header.** The leading part of a message that contains information such as the source or destination code of the message, the message priority, and the type of message. See also *header.*

**message lock mode.** A type of lock mode in which an external logical unit (LU) is in lock mode for the duration of a single inquiry and reply. Contrast with *extended lock mode.* See also *station lock.*

**message priority.** The order in which messages in a destination queue are transmitted to a destination. Higher-priority messages are forwarded before lower-priority messages. See also *route transmission priority, station transmission priority.*

**message queue data set.** A TCAM data set that contains one or more destination queues. A message queue data set contains messages that have been processed by the incoming group of a message handler and are waiting for TCAM to dequeue them, route them through an outgoing group of a message handler, and send them to their destinations. Up to three message queue data sets (one in main storage, one on reusable disk, and one on nonreusable disk) may be specified for a TCAM message control program.

**message routing.** A message control program (MCP) function that determines the correct destination for each message received by the MCP and places the message on the appropriate destination queue. See *affinity-based routing, invariant routing, transaction-based routing.* See also *routing by destination, routing by key.*

**message segment.** The portion of a message that is contained in a single request unit (RU).

**message text.** Synonym for *text.*

**message unit.** In SNA, a general term for the unit of data processed by any layer; for example, a basic information unit, a path information unit, or a request/response unit (RU).

**MH.** Message handler.

**mode name.** The name of an entry in the logon mode table.

**multiple-domain network.** A network with more than one system services control point (SSCP). Contrast with *single-domain network.*

**Multiple Virtual Storage (MVS).** An IBM program product whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for System/370. It is a software operating system controlling the execution of programs.

**MVS.** Multiple Virtual Storage operating system.

**NAU.** Network addressable unit.

**NAU services.** In SNA, the functions provided by the NAU services manager layer and the FMD services layer.

**NCP.** (1) Advanced Communications Function for the Network Control Program. An IBM program product that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. (2) A general term for a program that is generated by the user from a library of IBM-supplied modules and controls the operation of a communication controller.

**negative response.** In SNA, a response indicating that a request did not arrive successfully or was not processed successfully by the receiver. Contrast with *positive response.* See *exception response.*

**negotiable bind.** In SNA, a function that allows two LU-LU half-sessions to negotiate the parameters of a session when the session is being activated.

**network.** In data processing, a user application network. See *public network, SNA network, user application network.*

**network address.** In SNA, an address, consisting of subarea and element fields, that identifies a link, a link station, or a network addressable unit (NAU). Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See *local address, TCAM network address.* See also *network name.*

**network addressable unit.** In SNA, a logical unit (LU), a physical unit (PU), or a system services control point (SSCP).

**network control program.** A program, generated by the user from a library of IBM-supplied modules, that controls the operation of a communication controller.

**Network Control Program (NCP).** An IBM program product that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. Officially, the Advanced Communications Function for the Network Control Program.

**network operator.** In SNA, a person or program responsible for controlling the operation of all or part of a network.

**network services procedure error (NSPE).** A request unit that is sent by a system services control point (SSCP) to a logical unit (LU) when a procedure requested by that LU has failed.

**Network Terminal Option (NTO).** An IBM program product that allows certain non-SNA devices to participate in sessions with SNA application programs in the host processor. NTO converts non-SNA protocol to SNA protocol when data is sent to the host from a non-SNA device and reconverts SNA protocol to non-SNA protocol when data is sent back to the device.

**networking.** In a multiple-domain network, communication between domains. See *extended networking.*

**NIB.** Node initialization block.

**no response.** In SNA, a value in the form-of-response-requested field of the request header (RH) indicating that no response is to be returned to the request, whether or not the request is received and processed successfully. Contrast with *definite response, exception response.*

**node.** In SNA, a junction point in a network that contains a physical unit (PU). A node contains network addressable units, path control components, and may contain boundary function. See *TCAM node.*

**node identifier.** That portion of the TCAM network address of a resource that indicates which

TCAM node provides the message queuing for that resource. See *resource identifier*.

**node initialization block (NIB).** In VTAM, a control block associated with a particular node or session that contains information used by the VTAM application program to identify the node or session and to indicate how communication requests on a session are to be handled by VTAM.

**node table.** For TCAM extended networking, a main-storage table that associates each node identifier with internodal destination queues.

**non-SNA terminal.** A terminal that supports non-SNA protocols; for example, channel-attached 3270 Information Display System or devices supported by Network Terminal Option (NTO) that use binary synchronous protocols. Contrast with *SNA terminal*.

**normal flow.** In SNA, a data flow that is used for most requests and responses. The expedited flow is independent of and used to control the normal flow. Requests and responses on a normal or expedited flow are processed sequentially within the path, but the expedited flow traffic may be moved ahead of the normal flow traffic within the path. Contrast with *expedited flow*.

**NSPE.** Network services procedure error.

**NTO.** Network Terminal Option.

**OEF.** Origin element field.

**online retrieval.** The function of a system service program that allows system operators to retrieve disk-queued messages based upon origin or destination, time of entry, or input or output sequence number.

**OPCE.** Operator control element.

**open subroutine.** A subroutine of which a replica must be inserted at each place in a computer program at which the subroutine is used. Contrast with *closed routine*.

**operator command.** Synonym for *operator control command*.

**operator control.** Synonym for *basic operator control, extended operator control*.

**operator control command.** A command entered from an operator control station to examine or alter the status of the TCAM system during execution of TCAM.

**operator control element (OPCE).** A unit assigned to each operator control command that is used by the operator control routines to process the command.

**operator control station.** Synonym for *basic operator control station, basic primary operator control station, basic secondary operator control station, extended operator control station, extended primary operator control station, extended secondary operator control station*.

**option field.** A storage area containing data relating to a particular external logical unit (LU) or application program. Certain message-handler routines that need origin- or destination-related data to perform their functions have access to data in an option field. User-written message-handler exit routines also have access to data in an option field.

**option table.** A table that contains option fields of user-provided information, using certain TCAM macro instructions, related to external logical units (LUs) or application programs.

**origin.** An external logical unit (LU) or application program from which a message or other data originates. See also *destination*.

**outbuffer subgroup.** The part of a message handler (MH) outgoing group that operates on each segment of an outgoing message.

**outgoing group.** That portion of the message handler that handles messages sent from the message control program (MCP) to any external logical units (LUs) or application programs. Contrast with *incoming group*.

**outheader subgroup.** The part of a message handler (MH) outgoing group that operates on all o part of an outgoing message header.

**outmessage subgroup.** The part of a message handler (MH) outgoing group that specifies actions to be taken after the entire message has been sent to an external logical unit (LU), or when special processing or error conditions are detected.

**output data set.** A data set that contains the messages or records returned from an application program to the message control program by a process entry in the terminal table. Contrast with *input data set*.

**path switch.** A field in the option table that determines whether a given subgroup is to be executed for a message.

**PCB.** Process control block.

**physical message.** The data entered on a link during a complete transmission sequence, from the first byte of data to the end of the transmission character. Contrast with *logical message.* In SNA, synonym for *RU chain.*

**PLU.** Primary logical unit.

**POF restart.** Point-of-failure restart.

**point-of-failure (POF) restart.** A type of warm restart of the message control program (MCP) that uses incident records to update an environment record when the system is restarted following closedown or system failure. Contrast with *point-of-last-environment restart.* See *cold restart, warm restart.*

**point-of-last-environment (POLE) restart.** A type of warm restart of the message control program (MCP) that ignores incident records when the system is restarted following closedown or system failure. Contrast with *point-of-failure restart.* See *cold restart, warm restart.*

**POLE restart.** Point-of-last-environment restart.

**positive response.** In SNA, a response unit that indicates that a request was successfully received and processed. Contrast with *exception response, negative response.*

**prefix.** Synonym for *buffer prefix.*

**presentation services.** Synonym for *session presentation services.*

**primary end of a session.** Deprecated term for *primary half-session.*

**primary half-session.** The half-session that sends the session-activation request. Contrast with *secondary half-session.* See also *primary logical unit.*

**primary logical unit (PLU).** In SNA, a logical unit that contains the primary half-session for a particular LU-LU session. A PLU issues a Bind Session command to establish an LU-LU session. Contrast with *secondary logical unit.* See also *logical unit.*

**primary operator control station.** See *basic primary operator control station, extended primary operator control station.*

**priority.** Synonym for *message priority, station transmission priority.*

**process control block (PCB).** A message control program (MCP) data area that is necessary for communication between the MCP and an application program.

**process entry.** A terminal-table entry that represents an application program. One entry must be defined for each queue to which an application program can issue a GET or READ macro instruction, and at least one entry must be defined for all PUT and WRITE macro instructions issued from the same application program.

**process queue.** A destination queue for an application program. See *destination queue.*

**protocol.** In SNA, the meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components. SDLC, BSC, and start-stop (SS) are link protocols.

**public network.** A network established and operated by communication common carriers or telecommunication Administrations for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public. Contrast with *user application network.*

**QCB.** Queue control block.

**queue.** (1) A line or list formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted in a message-routing system. (2) To arrange in, or form a queue. See also *queuing.*

**queue control block (QCB).** A control block that is used to regulate the sequential use of a programmer-defined facility among requesting tasks.

**queuing.** The programming technique used to handle messages that are awaiting delivery. See also *queue.*

**quick closedown.** A closedown in which message traffic is stopped as soon as any messages in the process of being sent or received at the time the request for closedown is received are transmitted. Contrast with *flush closedown.*

**read-ahead queue.** An area of main storage from which an application program obtains work units in advance of their being requested by the application program.

**record.** A collection of related data or words, treated as a unit; for example, in stock control, each invoice could constitute one record.

**reply.** (1) In TCAM, response to an inquiry. (2) In SNA, a request unit sent only in reaction to a received request unit. For example, Quiesce Complete is the reply sent after receipt of Quiesce at End of Chain.

**request.** In SNA, a message unit that signals initiation of a particular action or protocol. For example, Initiate Self is a request for activation of an LU-LU session.

**request header (RH).** In SNA, an RU header that precedes a request unit.

**request/response header (RH).** In SNA, control information preceding a request/response unit (RU) that specifies the type of RU (request unit or response unit) and contains control information associated with that RU. See also *request/response unit, connection point manager*.

**request/response unit (RU).** In SNA, a general term for a request unit or a response unit.

**request parameter list (RPL).** In VTAM, a control block that contains the parameters necessary for processing a request for data transfer, for establishing or terminating a session, or for some other operation.

**request unit (RU).** In SNA, a message unit that contains control information such as a request code or function management (FM) headers, end-user data, or both.

**resource identifier.** That portion of the TCAM network address of a resource that uniquely identifies the resource within the message control program (MCP) providing the message queuing for that resource. See also *node identifier*.

**resource management block (RMB).** A collection of control blocks all of which are associated with a particular external logical unit (LU). The RMB contains a line control block (LCB), a station control block (SCB), a savearea/workarea (SAU), and a request parameter list (RPL).

**resource table.** In TCAM extended networking, a main-storage table that associates each resource identifier with an external logical unit (LU) or application program.

**response.** In SNA, a message unit that acknowledges receipt of a request. A response consists of a response header, a response unit, or both.

**response header (RH).** A header, optionally followed by a response unit, that indicates whether the response is positive or negative and may contain a pacing response. See also *isolated pacing response, negative response, pacing response, positive response.*

**response unit (RU).** In SNA, a message unit that acknowledges a request unit; it may contain prefix information received in a request unit. If it is positive, the response unit may contain additional information (such as session parameters in response to a Bind Session request). If it is negative, the response unit may contain sense data that defines the exception condition.

**RH.** Request/response header.

**RMB.** Resource management block.

**routing.** Synonym for *message routing.*

**routing affinity.** A temporary relationship between a source and a destination.

**routing by destination.** Message routing based upon a destination name. Contrast with *routing by key.*

**routing by key.** Message routing based upon a key, which matches a definition in the key table. The key identifies the destination of a message or special processing to be done on that message. Contrast with *routing by destination.*

**routing key.** Synonym for *key.*

**routing key table.** Synonym for *key table.*

**RPL.** Request parameter list.

**RU.** Request/response unit.

**RU chain.** In SNA, a set of related request units (RU) that are consecutively transmitted on a particular normal or expedited data flow. The request unit (RU) chain is the unit of recovery. If one of the request units (RUs) in the chain cannot be processed, the entire chain is discarded.

**save/restore message queues (SMQ).** The function of a system service program that saves unsent messages on sequential storage devices and restores them to an altered message control program (MCP) following a cold restart. This program also assists in recovery when the message queue data set on nonreusable disk becomes full. The program may be used to obtain an online dump of unsent messages from one or more destination queues on disk.

**scan pointer.** A pointer that refers to the proper header field when the macro instruction that acts upon that field is given control. Some user-specified macro instructions use this pointer to locate the field on which they act and automatically move the pointer to the next field before passing control to the next macro instruction. The user must be aware of positioning of the scan pointer when designing the message handler.

**SDLC.** Synchronous data link control.

**secondary end of a session.** Synonym for *secondary half-session.*

**secondary half-session.** In SNA, the half-session that receives the session-activation request. Contrast with *primary half-session.* See also *secondary logical unit.*

**secondary logical unit (SLU).** In SNA, the logical unit that contains the secondary half-session for a particular LU-LU session. Contrast with *primary logical unit.*

**secondary operator control station.** Synonym for *basic secondary operator control station, extended secondary operator control station.*

**segment.** A portion of a message that can be contained in a buffer. See *BIU segment.*

**selective cryptographic session.** A cryptographic session in which an application program is allowed to specify the request units to be enciphered. Contrast with *mandatory cryptographic session.*

**service facility.** An auxiliary routine that runs under control of the message control program (MCP) and is invoked when needed by user code in the MCP. on an as-needed basis. Contrast with *system service program, utility.*

**session.** In SNA, a logical connection between two network addressable units that can be activated, tailored to provide various protocols, and deactivated, as requested. The session-activation request and response can determine options relating to the rate and currency of data exchange, the control of contention and error recovery, and the characteristics of the data stream. Sessions compete for network resources such as the links within the path control network. See *half-session, LU-LU session.* See also *LU-LU session type, PU-PU flow.*

**session activation.** In SNA, the process of exchanging a session activation request and a positive response between network addressable units. Contrast with *session deactivation.* See also *start.*

**session-activation request.** In SNA, a request that activates a session between two network addressable units and specifies session parameters that control various protocols during session activity. Contrast with *session deactivation request.*

**session control.** (1) One of the components of transmission control. Session control is used to purge data flowing in a session after an unrecoverable error occurs, to resynchronize the data flow after such an error, and to perform cryptographic verification. (2) A request/response unit (RU) category used for requests and responses exchanged between the session control components of a session and for session activation or deactivation requests or responses.

**session count.** (1) The number of currently active LU-LU sessions for a particular logical unit. (2) The number of currently active sessions for a particular virtual route.

**session deactivation.** The process of exchanging a session-deactivation request between two network addressable units. Contrast with *session activation.* See also *stop.*

**session deactivation request.** A request that deactivates a session between two network addressable units. Contrast with *session activation request.*

**session end.** Synonym for *half-session.*

**session information block (SIB).** A control block that contains information about a particular SNA session.

**session initiation.** Synonym for *LU-LU session initiation.* See also *LU-LU session termination.*

**session-initiation request.** An initiate or logon request from a logical unit (LU) to a systems services control point (SSCP) so that an LU-LU session can be activated.

**session parameters.** In SNA, the parameters that specify or constrain the protocols (such as bracket protocol) for a session between two network addressable units. See also *logon mode.*

**session presentation services.** A component of the function management data (FMD) services layer that provides, within LU-LU sessions, services for the application programmer or terminal operator such as formatting data to be displayed or printed.

**session sequence number.** In SNA, a sequentially incremented identifier that is assigned by data flow control to each request unit on a particular normal flow of a session, typically an LU-LU session, and is checked by transmission control. The identifier is carried in the transmission header of the path information unit and is returned in the transmission header of any associated response.

**session termination.** Synonym for *LU-LU session termination.*

**SIB.** Session information block.

**single-domain network.** A network with one system services control point (SSCP). Contrast with *multiple-domain network.*

**single entry.** A terminal-table entry associated with a single external logical unit (LU) or application program. Contrast with *distribution entry.*

**SLU.** Secondary logical unit.

**SMQ.** Save/restore message queues.

**SNA.** Systems Network Architecture.

**SNA network.** The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It makes possible reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. An SNA network consists of network addressable units, boundary function components, and the path control network.

**SNA node.** A node that uses SNA protocols.

**SNA session.** A logical connection, established between two network addressable units (NAUs), to allow them to communicate. The session is uniquely identified by a pair of network addresses identifying the origin and destination NAUs of any transmissions exchanged during the session. See *LU-LU session, pseudo LU-LU session.*

**SNA terminal.** A terminal that supports SNA protocols. Contrast with *non-SNA terminal.*

**SSCP.** System services control point.

**SSP.** (1) In TCAM, a system service program. (2) Advanced Communications Function for the Systems Support Programs. An IBM product program made up of a collection of utilities and small programs. SSP is required for operation of the NCP.

**start.** For external logical units (LUs) in TCAM, the state in which an LU is able to enter an LU-LU session.

**start-stop (SS) transmission.** Asynchronous transmission in which a group of bits is preceded by a start bit that prepares the receiving mechanism for the reception and registration of a character and is followed by at least one stop bit that enables the receiving mechanism to come to an idle condition pending the reception of the next character. Contrast with *binary synchronous transmission, synchronous data link control.*

**startup/restart message generation facility.** A TCAM service facility that generates and sends tailored messages to external logical units (LUs) when the message control program (MCP) is started or restarted.

**station.** One or more terminals or devices at a particular location; for example, an external logical unit (LU).

**station lock.** A facility that maintains a connection between a station and an application program to ensure that the next message received by the station, after it enters an inquiry message, is a reply to that inquiry. See also *extended lock mode, lock mode, message lock mode.*

**station transmission priority.** The relative order of the host sending and receiving messages. Host sending has priority over host receiving. See *message priority.*

**stop.** In TCAM, the state in which a logical unit (LU) is not able to enter an LU-LU session. This state also terminates any existing LU-LU sessions involving that LU.

**symbol.** In assembler language, a character or character string that represents addresses or arbitrary values. A symbol must meet the following requirements: (a) A symbol may consist of no more than eight characters, the first character being a letter (A through Z, $, #, or @) and the other characters being either letters or digits. (b) No blanks or special characters are allowed in a symbol.

**synchronous data link control (SDLC).** A discipline conforming to subsets of the Advanced Data Communication Control Procedure (ADDCP) of the American National Standards Institute and High-Level Data Link Control (HDLC) of the International Standards Organization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection.

Transmission exchanges may be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection may be point-to-point, multipoint, or loop. Contrast with *binary synchronous transmission, start-stop transmission.*

**system service program (SSP).** An IBM-supplied or user-supplied program that performs system-oriented auxiliary functions in support of the MCP. System service programs run under the control of the initiator as attached subtasks. Contrast with *service facility, utility.* See also *basic operator control, extended operator control, online retrieval, save/restore message queues, internodal awareness, internodal sequence number synchronization.*

Note: *The abbreviation SSP has two references. See also SSP.*

**system services control point (SSCP).** In SNA, a focal point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network. Multiple SSCPs, cooperating as peers, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its domain.

**Systems Network Architecture (SNA).** The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

**TCAM.** Advanced Communications Function for TCAM. An IBM program product that provides queued message handling. TCAM, Versions 1 and 2, are telecommunications access methods, but TCAM, Version 3, is a message handling subsystem.

**TCAM application program.** A program that is user written and interfaces with the message control program (MCP) using READ, WRITE, CHECK, GET, or PUT macro instructions.

**TCAM destination address field (TDAF).** A field in the fixed header prefix of a message that contains the TCAM network address of the destination of the message. Contrast with *TCAM origin address field.*

**TCAM host logical unit (LU).** A TCAM-generated logical unit (LU) that is the access method control block (ACB) interface to VTAM, for example, PROGID. External LUs must establish a session with a TCAM host LU in order to use TCAM services. See *host logical unit (LU).*

**TCAM network address.** A unique identifier for an application program or an external logical unit (LU) in an extended networking environment. A TCAM network address consists of a node identifier and a resource identifier. See also *node identifier, resource identifier.*

**TCAM node.** A message control program (MCP) to which there has been assigned a node identifier. See also *node identifier.*

**TCAM origin address field (TOAF).** A field in the fixed header prefix of a message that contains the TCAM network address of the originator of the message. Contrast with *TCAM destination address field.*

**TCAM subtask table (TST).** A table containing entries for programs eligible to run as initiator subtasks.

**TCAM system.** A subsystem controlled by a single message control program (MCP) with a collection of external logical units (LUs) and application programs.

**TDAF.** TCAM destination address field.

**terminal table.** In TCAM, an ordered collection of information about each origin or destination of messages in the network. See also *terminal-table entry.*

**terminal-table entry (TTE).** The information in the terminal table that identifies each origin or destination of messages in the network. See *cascade entry, logtype entry, process entry, single entry.*

**termination.** Synonym for *LU-LU session termination.*

**text.** That part of the message that is not the header or control information.

**text buffer.** A buffer containing any segment of a message other than the first segment, which is contained in a header buffer. Contrast with *header buffer.*

**TOAF.** TCAM origin address field.

**transaction-based routing.** Message routing in which messages are routed to their destinations individually, according to one or more destination names or routing keys entered in the message header by the originator. See *affinity-based routing,* *invariant routing.* See also *routing by destination, routing by key.*

**transmission category.** In TCAM extended networking, utility sessions. All messages in the same transmission category have similar characteristics and should be handled similarly. For example, messages flowing in an inquiry/reply application and messages flowing in a high-volume, low-priority data collection application are placed in different transmission categories. Different versions of the following TCAM techniques and capabilities may be applied to messages in different transmission categories: queuing medium, message priority, sequence checking, error handling, load balancing, and data staging.

**transmission services profile.** In SNA, a specification in a session-activation request of transmission control protocols (such as session-level pacing and the usage of session-control requests) to be supported by a particular session. Each defined transmission services profile is identified by a number.

**TST.** TCAM subtask table.

**TTE.** Terminal-table entry.

**unit.** Synonym for *buffer unit, work unit.*

**user application network.** A configuration of data processing products, such as processors, controllers, and stations, established and operated by users for the purpose of data processing or information exchange, which may use services offered by common carriers or telecommunications Administrations. Contrast with *public network.*

**utility.** In TCAM, an auxiliary routine designed to support the message control program (MCP), which runs under the control of the operating system. Contrast with *system service program, service facility.*

**utility session.** In TCAM extended networking, a pair of LU-LU sessions between TCAM nodes. One utility session is established between each pair of TCAM nodes for each transmission category defined for the pair. Data messages being routed from TCAM node to TCAM node flow on the utility session corresponding to their transmission category.

**VTAM.** (1) Advanced Communications Function for VTAM, an IBM program product. (2) Virtual Telecommunication Access Method.

**VTAM application program.** A program that has opened an access method control block (ACB) to identify itself to VTAM and can now issue VTAM macro instructions. See *TCAM application program.*

**warm restart.** Restart of TCAM following either a quick or a flush closedown. The TCAM checkpoint/restart service facility restores the TCAM environment as nearly as possible to its condition before closedown or failure. Contrast with *cold restart.* See *point-of-failure restart, point-of-last-environment restart.*

**WATS.** Wide Area Telephone Service, which provides a special line on which the subscriber may make unlimited calls to certain zones on a direct-distance-dialing basis for a flat monthly charge.

**work area.** An area of storage related to an application program that receives messages or records transferred to the application program from TCAM by GET or READ macro instructions, and from which messages or records are transferred to TCAM by PUT or WRITE macro instructions.

**work unit.** The amount of data transferred from TCAM to an application program by a single GET or READ macro instruction or transferred from an application program to TCAM by a single PUT or WRITE macro instruction. A work unit may be a message or a record.

**zero-length buffer.** A buffer that is sent to the message handler to indicate that there is an error on the link. If user code does not execute correctly for a zero-length buffer, the programmer must check for zero length and branch around the code that does not execute correctly.

# Index

A

ADDR operand
    COREDSP macro 5-8
address operand
    POINT macro 5-33
ALTDEST operand 2-12
API (Application Program Interface)
    See Application Program Interface (API)
application program
    application message handler 1-2
    batch processing 1-4
    checkpoint coordination with TCAM 4-16
    coding 2-13
    components between TCAM and an 2-1
    coordinating MCP and application-program restarts 4-20
    coordinating MCP and restarts with 4-20
    DCB relationship 2-2
    dead-letter queue 4-2
    defining buffers A-1
    defining buffers for application programs A-1
    device message handler 1-2
    displaying main storage within an 4-24
    examples of message routing and processing (figure) 1-7
    in a network 1-2
    initialization in the MCP for basic operator control in an 4-7
    inquiry/reply facility 3-3
    inspecting and changing 4-10
    interactive processing 1-4
    interface with an MCP 1-7
    issuing operator commands from an 4-1
    issuing operator control commands 4-1
    MCP relationship 1-7
    message flow 3-1
    operator commands from an 4-1
    optional TCAM facilities 4-1
    starting 2-19
    starting TCAM 2-19
    stopping the MCP 2-20
    testing 2-17
    transferring data between TCAM and 3-1
    transferring data between TCAM and an 3-1
    work unit 3-1
Application Program Interface (API)
    checklist for operand coding for A-4
    components of 2-1
    components of the interface 2-19
    defining 2-1, 2-19
        parameters 2-1, 2-19
    description 1-7
    stopping and starting the API 2-19

B

basic access technique
    what macros used 3-22
basic operator command
    format 4-1
basic operator control
    system service program 4-1
buffer
    coding checklist A-4
    design A-1
buffers
    defining A-1
    GET, READ A-2
    PUT, WRITE A-2

C

CHECK macro 5-2
checklist for possible coding errors B-1
CLOSE macro 2-20
coding application programs 2-13
compatibility, MCP and an application program 1-7
completion codes for basic access 3-27
concurrent retrieval of multiple messages 3-31
control blocks
    application program 2-2, 2-3
        input DCB macro 2-3
    MCP (Message Control Program) 2-10
control elements
control operand, use for operator commands 4-8
controlchars, field of command 4-3
CONV operand
    COREDSP macro 5-9
converting numbers 4-26
CONVTO operand
    TCBINCNV macro 5-45
COREDSP macro
    use 5-8
CVT=based TCAM 2-9

D

data definition card parameters 2-18
data transfer macros 2-11, 3-22
DCB macro 2-6
  DCB macro (input) 5-15
  DCB macro (output) 5-22
  input operands 2-3
  output operands 2-4
dcbname operand
  CLOSE macro 5-6
  COREDSP macro 5-9
  GET macro 5-25
  OPEN macro 5-31
  POINT macro 5-33
  PUT macro 5-36
  QRESET macro 5-40
  READ macro 5-43
  WRITE macro 5-54
DDNAME operand 2-18
dead-letter queue 4-2
DECBNAME operand
  CHECK macro 5-2
  READ macro 5-43
  WRITE macro 5-54
defining buffers A-1
DELETE operand, TPDATE macro 5-52
destination field work area 3-15
destination QCB
  inspecting 4-13
disk data sets, retrieving messages 3-30
DISPLAY basic operator command, displaying group, line or status 4-4
DKJFND macro 5-24
DTSAREA operand
  TPDATE macro 5-52

E

end-of-file specifying 3-20
end-of-message specifying 3-21
ending operator commands 4-4
extended operator commands 4-2, 4-9
extended operator control 4-9
  system service program 4-10

F

FLUSH operand
  MCPCLOSE macro 5-28
format for basic operator commands 4-3
forwarding
  direct 4-9
  end-to-end sessions, using 4-9

G

GET buffers A-2
GET macro 5-25
  function 3-28

H

HALT basic operator command, system closedown 4-4
header control information 3-19
HOLD basic operator command, hold message on a queue 4-4

I

IEDOPCTL macro, use for basic operator commands 4-7
input data control block (DCB) macro 2-3
INPUT operand
  TCBINCNV macro 5-45
inquiry/reply facility 3-3
inspecting a QCB 4-13
inspecting TCAM control elements 4-10
Installation Procedures for
  Assembly, Linkedit, Execution 2-16
INTRO macro A-4
issuing basic operator control commands 4-5
issuing extended operator commands 4-9
issuing TCAM Operator Control Commands 4-1

J

job language control (JCL) considerations 2-18

K

KEYPROC macro, use for extended operator commands 4-10

L

length operand
    READ macro   5-43
    WRITE macro   5-54
LIMIT operand
    QCOPY macro   5-38

M

macro reference guide, application program   5-1
macros
    &I2@arean.
        CKREQ macro   5-5
        COREDSP macro   5-8
        DCB macro (input)   5-10
        DCB macro (output)   5-18
        DKJFND macro   5-24
        GET macro   5-25
        MCOUNT macro   5-27
        MCPCLOSE macro   5-28
        MRELEASE macro   5-30
        OPEN macro   5-31
        POINT macro   5-33
        PUT macro   5-36
        QCOPY macro   5-38
        QRESET macro   5-40
        READ macro   5-43
        TCBINCNV macro   5-45
        TCHNG macro   5-47
        TCOPY macro   5-50
        TPDATE macro   5-52
        WRITE macro   5-54
    application program
    CHECK macro   5-2
MAX operand
    QRESET macro   5-41
MCOUNT macro   5-27
    use   4-22
MCP (Message Control Program)
    See Message Control Program (MCP)
MCPCLOSE macro   2-20, 5-28
    closedown of the MCP   2-20
    function of the MCPCLOSE macro   2-20
    stopping the MCP   2-20
Message Control Program (MCP)
    control blocks in   2-10
    initializing MCP for basic operator control   4-7
    interface between application program and   1-7
message flow through TCAM   3-1
message handlers   1-1, 1-2
    incoming group   1-3
        destination queue   1-3
    outgoing group   1-3
        Read-Ahead queue   1-3
message handling considerations   3-19
message processing, specifying   3-5
message vs record processing   3-5
messages
    compatible format   1-8
    message queue dataset   1-4
    retransmitting and retrieving messages   3-34
    retrieving messages from disk   3-30
messages and records
    description   3-3
    processing differences   3-4
MF operand
    CLOSE macro   5-7
MODIFY basic operator command, modify polling list or option fields   4-4
MRLEASE macro   5-30
    use   4-23
multiple messages, concurrent retrieval   3-31
multiple TCAMs, special considerations   2-9

N

NAME operand

O

opening and closing TCAM-related DCBs   2-20
    CLOSE macro   2-20, 5-6
    OPEN macro   5-31
operand(s) field of command   4-4
operation field of command   4-4
operator control
    basic operator control, system service program   4-2
    extended operator control, system service program   4-9
OPTCD operand   2-6
    input data control block (DCB) macro   2-3
origin field, work area   3-15
outgoing group   1-3
OUTPUT operand
    TCBINCNV macro   5-45

P

PASSWRD operand
    MCPCLOSE   5-29
    MRELEASE macro   5-30
    TCHNG macro   5-47
PCB macro   2-11
    BUFSIZE operand, PCB macro   2-11
    checklist for coding application programs   A-4
    special considerations   2-11

planning TCAM
  application programs 1-1
  environment 1-1
POINT macro 5-33
  related operands 3-33
  retrieving messages 3-30
position field, work area 3-18
  SAM Prefix field 3-18
processing a message 3-5
processing a record 3-7
PUT buffers A-2
PUT macro 5-36
  avoiding simultaneous multiple execution 3-29
  function of 3-29
PUTCNT operand
  PCB macro 2-11

Q

QBACK operand
  TPROCESS macro 2-12
QCB (queue control block)
  See queue control block (QCB)
QCOPY macro 5-38
QNAME parameter
  jobname 2-9
  procname 2-9
QRESET macro 5-40
  retransmitting messages 3-29
queue control block (QCB)
  control element 4-10
queue, determining number on 4-22
queued access technique 3-28
QUEUES operand
  TPROCESS macro 2-12
QUICK operand
  MCPCLOSE macro 5-28

R

READ buffers A-2
READ macro 5-43
read-ahead queue A-2
RECDEL operand
  TPROCESS 2-12
RECDLM operand
  TPDATE operand 5-52
record
  record processing 3-4
RELEASE basic operator command, release messages from a queue 4-23
reserve bytes 3-32
Return Codes
  &I2@return.
    CKREQ macro 5-6
    CHECK macro 5-3
    COREDSP macro 5-10
    MCOUNT macro 5-28
    MCPCLOSE macro 5-29
    MRELEASE macro 5-31
    POINT macro 5-35
    PUT macro 5-37
    QCOPY macro 5-40
    READ macro 5-45
    TCHNG macro 5-49
    TCOPY macro 5-51
    TPDATE macro 5-54
    WRITE macro 5-56

S

S operand
  READ macro 5-43
  WRITE macro 5-54
SAM-Prefix field, work area 3-18
SECTERM operand
  TPROCESS macro 2-13
  use for operator commands 4-7
SECURE operand
  TPROCESS macro 2-13
SETEOF macro
  use 3-20
SF operand
  READ macro 5-43
  WRITE macro 5-54
SFLAG operand
  PCB macro 2-11
starting application programs 2-19
statname operand
  MRELEASE macro 5-30
  TCOPY macro 5-50
STOP operand
  DCB macro 2-6
stopping the API 2-19
SYNADAF message buffer format 2-9

T

TCAM operand
  DKJFND macro 5-24
TCBINCNV macro 5-45
TCHNG macro 5-47
TCOPY macro 5-50
TERMINAL macro, use for basic operator commands 4-8
terminal table DSECT length (figure) 4-13
terminal table entry
  changing 4-14
  inspecting 4-11
TERMNAME operand

QCOPY macro 5-38
TCHNG macro 5-47
time and date received 4-23
TIMEDLY operand
PCB macro 2-11
TPDATE macro 4-23
TPROCESS macro 2-12
checkpoint restart 4-21
special considerations 2-12
time and date 4-23
use for basic operator commands 4-8

W

work area
dynamic 3-14
optional fields 3-15
origin or destination field 3-15
position field 3-17
static 3-13
types 3-3
WORK operand
COREDSP macro 5-9
work unit
formats 3-9
size determination (figure) 3-8
WRITE buffers A-2
WRITE macro 3-24
avoiding simultaneous multiple execution 3-29
use for basic operator commands 4-8

**Advanced Communications Function for TCAM Version 3**

**READER'S COMMENT FORM**

**Application Programming (MVS)**

**Order No. SC30-3233-1**

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

**Note:** Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____________________________

____________________________

____________________________

____________________________

What is your occupation? ____________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC30-3233-1

**Reader's Comment Form**

**Fold and tape** **Please Do Not Staple** **Fold and tape**

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E03
P.O. Box 12195
Research Triangle Park, N.C. 27709-2195

**Fold and tape** **Please Do Not Staple** **Fold and tape**

IBM®