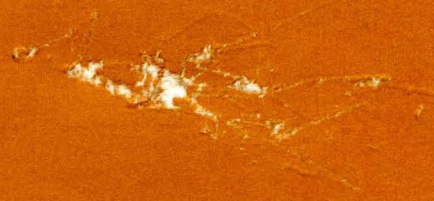
GC30-2024-0

IBM System/360 Operating System Telecommunications Access Method (TCAM) Programmer's Guide and Reference Manual

Program Number 360S - CQ - 548

Systems



IBM

GC30-2024-0

IBM System/360 Operating System Telecommunications Access Method (TCAM) Programmer's Guide and Reference Manual

Systems

Program Number 360S - CQ - 548

This book is a reference manual and coding guide for the programmer who must construct or modify a TCAM Message Control Program (MCP), or who must write a TCAM-compatible application program. It explains how to write a TCAM MCP, how to write a TCAM-compatible application program, and how to use a variety of auxiliary service facilities. Also included is information that might be of use in planning and setting up a teleprocessing system incorporating TCAM. The reader is expected to be familiar with the contents of the publication *IBM System/360 Operating System Telecommunications Access Method (TCAM) Concepts and Facilities*, Order no. GC30-2022.

IBM

Preface

The first section of this book, *How to Use This Book* defines the audience for which this programmer's guide and reference manual is intended, explains how the book is organized, and suggests how the reader might best familiarize himself with its contents. The chart below lists alphabetically the key words that are used throughout the book to refer to other publications; accompanying the key words are the corresponding title and order number to which the key words refer.

Key Words Used in This Publication	*Title*	*Order No.*
Assembler Language	IBM System/360 Operating System Assembler Language	GC28-6514
Checkpoint/Restart Planning Guide	IBM System/360 Operating System Advanced Checkpoint/Restart Planning Guide	GC28-6708
Data Management Services	IBM System/360 Operating System Data Management Services	GC26-3746
Job Control Language	IBM System/360 Operating System Job Control Language	GC28-6539
Messages and Codes	IBM System/360 Operating System Messages and Codes	GC28-6631
Operator's Guide	IBM System/360 Operating System Operator's Guide	GC28-6540
Principles of Operation	IBM System/360 Operating System Principles of Operation	GA22-6821
Programmer's Guide to Debugging	IBM System/360 Operating System Programmer's Guide to Debugging	GC28-6670
Supervisor and Data Management Macro Instructions	IBM System/360 Operating System Supervisor and Data Management Macro Instructions	GC28-6647
Supervisor Services	IBM System/360 Operating System Supervisor Services	GC28-6646
System Generation	IBM System/360 Operating System System Generation	GC28-6554
TCAM Concepts and Facilities	IBM System/360 Operating System Telecommunications Access Method (TCAM) Concepts and Facilities	GC30-2022
TCAM PLM	IBM System/360 Operating System Telecommunications Access Method (TCAM) Program Logic Manual	GY30-2029
TOTE and Configuration Reference Manual	IBM Diagnostic Program TOTE/Configurator Users' Manual (Part No. 5172900)	
Utilities	IBM System/360 Operating System Utilities	GC28-6586

First Edition (January 1971)
This publication corresponds to Release 20.0 of IBM System/360 Operating System until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication with IBM systems or equipment, refer to the latest SRL Newsletter for editions that are applicable and current.

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

This manual has been prepared by the IBM Systems Development Division, Publications Center, Department E01, P. O. Box 12275, Research Triangle Park, North Carolina 27709. A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be sent to the above address.

Contents

How to Use This Book xi

Writing the Message Control Program 13
What the Message Control Program is 13
Functions of the MCP 13
User Tasks in Writing the MCP 13

Defining Terminal and Line Control Areas 15
Terminology 15
Line Control 15
Establishing Contact 17
Invitation 18
Constructing the Invitation List 18
INVLIST Macro Instruction 19
Nonswitched point-to-point or multipoint lines to stations using Polling Characters 21
Switched Lines To Terminals using Polling Characters 22
Switched Lines to Stations using ID Sequences 22
Switched or Nonswitched Contention Lines to Terminals which are not assigned ID Sequences 23
Output-only Lines to Stations Having no ID Sequences Assigned to Them 24
Selection 24
Constructing the Terminal Table 24
TTABLE Macro Instruction 25
OPTION Macro Instruction 26
TERMINAL Macro Instruction 29
Coding the TERMINAL Macro for a Component 36
Coding the TERMINAL Macro for a Line 37
TLIST Macro Instruction 40
TPROCESS Macro Instruction 41
LOGTYPE Macro Instruction 44
Maintaining Orderly Message Flow 45
Message Priority and Queuing 46
Transmission Priority 48
Transmission Priority for Nonswitched Polled Stations 48
Transmission Priority for Nonswitched Polled Stations Using TCAM's Buffering Feature 49
Transmission Priority for Nonswitched Contention Stations 50
Transmission Priority Switched Stations 50
Calls between the Computer and a Switched Station 51
The System Interval 53

Defining Buffers 55
Structure of a Buffer 55
The Buffer Unit Pool 57
Buffer Definition Checklist 59
Design Considerations 61
Size of Buffers 61
Number of Units 61
Size of Units 62
Dynamic and Static Buffer Allocation 62
Initial and Maximum Number of Buffers per Line 63
Other Buffer Design Considerations 63

Defining the MCP Data Sets 65
Line Group Data Sets 65
Characteristics of a Line Group 65
Creating a Line Group Data Set 65
Line Group DCB Macro Instruction 65
DD Statements for a Line Group 70

Message Queues Data Sets 72
Disk Queuing 72
Advantages and Disadvantages of Disk Queuing 73
Specifying Channel Program Blocks 74
How to Determine if too Many CPBs were Specified on the CPB = Operand of the INTRO Macro Instruction 75
How to Determine if too Few CPBs were Specified on the CPB = Operand of the INTRO Macro Instruction 75
Preformatting DASD Message Queues Data Sets 75
Using Multiple Arm Support 75
Reusable Disk Queues 76
Nonreusable Disk Queues 79
Main-storage Queuing 79
Specifying One or More Queuing Methods 81
Message Queues DCB Macro Instruction 83
DD Statements for Message Queues Data Sets 84
Checkpoint Data Set 85
Checkpoint DCB macro Instruction 85
DD Statement for the Checkpoint Data Set 86
Log Data Sets 87
User ABEND Exits 87

Activating and Deactivating the Message Control Program 91
Starting and Restarting TCAM 91
Initialization and Activation 91
INTRO Macro Instruction 92
OPEN Macro Instruction 101
READY Macro Instruction 104
Deactivation 105
Types of Closedown 105
Deactivating a TCAM System without Application Programs 105
Deactivating a TCAM System with Application Programs 106
CLOSE Macro Instruction 106
Sample MCP Activation and Deactivation Section 107

Designing the Message Handler 109
Message Format 109
The Message Header 110
Structure of the Message Handler 111
Selecting Message-handler Functions 115
Message Editing 115
Validity Checking 116
Message Routing 116
Record Keeping 116
Error Handling 116
System Control 117
Function Modification 118
Functions Provided by Delimiter Macros 118
Order of Macro Specification 118
The Scan Pointer 118
Message Flow through a Message Handler 122
Message Flow within an MH Group 123
Multiple Buffer Header Handling 124
Variable Processing within a Message Handler 129
Conditional Execution of Message Handler Functional Macros 129
User Code in a Message Handler 130
General Requirements and Restrictions 131
Multiple-buffer Header Considerations 131
Including an Open Subroutine 132
Including a Closed Subroutine 132
Using LOCOPT to Locate an Option Field 132
Using SETSCAN to Locate a Header Field 132
Using MSGTYPE to Locate a Header Field 134
Using the PARM parameter of the EXEC Job-control Statement 136
Message Handler Macro Return Codes 136

Message Translation . . . 137
TCAM's Hold/Release Facility . . . 139
Design Steps . . . 140
Delimiter Macro Instructions . . . 141
STARTMH . . . 142
INHDR . . . 145
INBUF . . . 146
INMSG . . . 147
INEND . . . 147
OUTHDR . . . 148
OUTBUF . . . 148
OUTMSG . . . 149
OUTEND . . . 149
Functional Macro Instructions . . . 150
CANCELMG . . . 150
CHECKPT . . . 151
CODE . . . 151
COUNTER . . . 154
CUTOFF . . . 155
DATETIME . . . 155
ERRORMSG . . . 157
FORWARD . . . 159
HOLD . . . 162
INITIATE . . . 163
LOCK . . . 165
LOCOPT . . . 167
LOG . . . 167
MSGEDIT . . . 168
MSGFORM . . . 179
MSGGEN . . . 180
MSGLIMIT . . . 182
MSGTYPE . . . 183
ORIGIN . . . 185
PATH . . . 187
PRIORITY . . . 190
REDIRECT . . . 192
SCREEN . . . 194
SEQUENCE . . . 196
SETEOF . . . 198
SETSCAN . . . 199
TERRSET . . . 202
UNLOCK . . . 202

Putting the MCP Together . . . 205
Arranging the Sections of the MCP . . . 205
Assembling, Linkage-editing, and Executing the Message Control Program . . . 206
Assembling an MCP . . . 206
Linkage-editing an MCP . . . 206
Executing an MCP . . . 206
Starting the MCP by a Catalogued Procedure . . . 207
Sample MCPs . . . 208
Message Switching Between Terminal Types . . . 208
Inquiry and Rapid Response . . . 213
File Updating with Checkpoint Coordination . . . 220

Writing TCAM-Compatible Application Programs . . . 233
Message Flow to an Application Program . . . 235
Overview of the MCP/Application-program Interface . . . 235
Defining the Components of the Interface . . . 236
Defining the Application Program Data Sets and the Process Control Block . . . 237
Input DCB Macro Instruction . . . 238
Output DCB Macro Instruction . . . 242
DD Statements for the Input and Output Data Sets . . . 245
PCB Macro Instruction . . . 245
Defining Buffers for the Application Program . . . 247

Defining Application-program Buffers . . . 247
Application-program Buffer Design Considerations . . . 247
Activating and Deactivating the Application-program Interface . . . 249
OPEN Macro Instruction for the Application Program . . . 249
CLOSE Macro Instruction for the Application Program . . . 251
MCPCLOSE Macro Instruction . . . 251
Transferring Data Between an MCP and an Application Program . . . 252
Defining the Application-program Work Area . . . 253
Static Work-area Definition . . . 253
Dynamic Work-area Definition . . . 254
Moving Data between Input and Output Work Areas . . . 254
Defining Optional Fields in the Work Area . . . 254
Origin and Destination Fields . . . 255
Position Field . . . 255
SAM Prefix . . . 256
Specifying Application-program Work Units . . . 257
Work-unit Formats . . . 257
Work-unit Types . . . 259
Processing the Message as a Work Unit . . . 259
Processing the Record as a Work Unit . . . 260
Signaling End of File and End of Message . . . 262
Coding TCAM's Data Transfer Macros . . . 263
GET Macro Instruction (QSAM only) . . . 264
PUT Macro Instruction (QSAM only) . . . 265
READ Macro Instruction (BSAM only) . . . 265
WRITE Macro Instruction (BSAM only) . . . 267
BSAM/TCAM Completion Codes . . . 269
CHECK Macro Instruction (BSAM only) . . . 269
Multiple-wait Capability . . . 270
Application Program Error Exits . . . 271
Input to the SYNAD Routine . . . 272
SYNADAF . . . 273
Network Control Facilities . . . 273
TCOPY Macro Instruction . . . 274
ICOPY Macro Instruction . . . 276
QCOPY Macro Instruction . . . 280
TCHNG Macro Instruction . . . 281
ICHNG Macro Instruction . . . 282
MRELEASE Macro Instruction . . . 284
TCAM's Message Retrieval Facility . . . 285
POINT Macro Instruction . . . 285
TCAM's Inquiry/Rapid Response Facility . . . 286
TCAM/SAM Compatibility . . . 289
Coordinating TCAM Checkpoints of the MCP with OS Checkpoints of the Application Program . . . 289
Using the CKREQ Macro Instruction for Coordination . . . 290
Suggestions for Using CKREQ . . . 291
Using the DCB Exit for Coordination . . . 292
Coordinating MCP and Application-program Restarts . . . 293

Using TCAM Service Facilities . . . 295
Operator Control . . . 295
Initialization for Operator Control . . . 295
General Format of Operator Commands . . . 295
Specifying Operator Commands . . . 297
Entering Operator Commands from an Application Program . . . 299
Incorrect Messages . . . 299
Operator Commands . . . 300
Checkpointing of Operator Commands . . . 326
TCAM I/O Error-recovery Procedures . . . 326
TCAM I/O Error-recording Facility . . . 327
Kinds of TCAM I/O Error Records . . . 327
Intensive-mode Error Recording . . . 328
Operator Awareness Message . . . 329
Gaining Access to Error Records . . . 329

Network Reconfiguration . . . 329
By Operator Commands . . . 329
By Application Program Macros . . . 330
TCAM Checkpoint/Restart Facility . . . 330
How the TCAM Checkpoint Facility Works . . . 333
Types of Checkpoint Record . . . 333
Scanning the Message Queues . . . 335
How to get the TCAM Checkpoint Facility . . . 338
Types of TCAM Restart . . . 339
Using TCAM's Message Logging Facility . . . 341
Uses of Message Logging . . . 341
How Message Logging Works . . . 342
How to Set Up a Message Logging Facility . . . 342
Debugging Aids . . . 344
Cross-reference Table . . . 345
TCAM Line I/O Interrupt Trace Table . . . 345
Writing on a Data Set for Later Printing . . . 346
COMWRITE Requirements and Format . . . 347
Dispatcher Subtask Trace Table . . . 347
Buffer Dump . . . 349
Writing Line Trace, STCB Trace, and Buffers to Disk Data Set . . . 349
COMEDIT Printing Utility . . . 352
Message Queues Data Set Dump . . . 353
On-line Test Function . . . 356
Advantages of TOTE . . . 356
Devices Supported . . . 356
TOTE Facilities . . . 356
System Requirements . . . 357
Main-storage Requirements . . . 357
TOTE Requirements . . . 357
Coding Requirements . . . 358
OS/SYSGEN Requirements . . . 358
JCL Requirements for TOTE/OLTs . . . 358

System Preparation . . . 361
Machine and Device Requirements . . . 361
Control Units and Terminal Types Supported . . . 361
Multiprocessing System . . . 361
System Generation Considerations . . . 364
Preformatting DASD Message Queues Data Sets . . . 365

Appendix A: TCAM Macro Formats . . . 367
Conventions Used . . . 367

Appendix B: Message Error Record . . . 369

Appendix C: How to make Transient Checkpoint and Operator Control Modules Resident . . . 373

Appendix D: Internal and Transmission Code Charts . . . 375

Appendix E: Running QTAM Application Programs under TCAM . . . 403

Appendix F: Summary of Operator Commands Classified by Operation . . . 405

Appendix G: Device Dependent Considerations . . . 407
Start/Stop Devices . . . 407
Binary Synchronous Communication (BSC) Terminals . . . 413
The TPEDIT Macro Instruction for the IBM 50 Magnetic Data Inscriber . . . 417

Appendix H: Conserving Main Storage . . . 427

Glossary . . . 429

Index . . . 437

Figures

Figure 1. Chart for Deciding Whether a TERMINAL Macro Should be Coded for a Switched Line . . . 38
Figure 2. Two Buffers Assigned to a Line Group; KEYLEN = 60 and BUFSIZE = 120 . . . 56
Figure 3. Unit Allocation when Main-storage Queuing (with or without Backup on Disk) is Specified . . . 58
Figure 4. Unit Allocation when Disk-only Queuing is Specified . . . 59
Figure 5. 706-byte Data Movement Resulting from Size Disparity between Input and Output Buffers . . . 64
Figure 6. Relative Record Numbers of Disk Message Queues Data Set Assigned Across Three Volumes . . . 73
Figure 7. Reorganizing a Reusable Data Set . . . 76
Figure 8. Sample MCP Activation and Deactivation Section . . . 108
Figure 9. Sample Format for an Incoming Message . . . 111
Figure 10. Sample Format for an Outgoing Message . . . 111
Figure 11. MH Subgroups and Macros . . . 114
Figure 12. Scan Pointer Movement . . . 119
Figure 13. Message Flow for a Switched Message . . . 122
Figure 14. Message Flow for a Message that is Processed by an Application Program . . . 123
Figure 15. Flow of a Two-segment Message with a Single-buffer Header through an MH . . . 125
Figure 16. Flow of a Two-segment Message with a Multiple-buffer Header through an MH . . . 126
Figure 17. Activation of a Closed, User-written Subroutine . . . 133
Figure 18. Deletion of Data from a Message Segment followed by Contraction of the Segment; KEYLEN = 60 and BUFSIZE = 120 . . . 178
Figure 19. Example of Using the MSGTYPE Macro Instruction . . . 185
Figure 20. Example of Using the PATH Macro Instruction to Vary MH Processing . . . 189
Figure 21. Example of Using the PRIORITY Macro Instruction . . . 192
Figure 22. Example of Inserting Line Address . . . 196
Figure 23. Sample Message-switching Program (3 parts) . . . 210
Figure 24. Sample Inquiry/Response Program (6 parts) . . . 214
Figure 25. Sample Checkpoint Coordination Program (10 parts) . . . 222
Figure 26. Interface between the Application Program and the MCP . . . 237
Figure 27. Relative Positions of Optional Fields in the Work Area . . . 257
Figure 28. Effect of a Work-unit's Type and Format on the Way in which TCAM Determines its Size . . . 263
Figure 29. Example of Multiple-wait Capability . . . 271
Figure 30. Terminal Table DSECT for Single, Line, and Group Entries . . . 275
Figure 31. Sample Invitation List Containing Three Entries . . . 277
Figure 32. Example of Using the CKREQ Macro Instruction for Checkpoint Coordination . . . 291
Figure 33. Operator Commands Classified by Areas Affected . . . 325
Figure 34. Formulas for Determining the Size of the Checkpoint Data Set (2 parts) . . . 338
Figure 35. Information Flow for Message Logging . . . 343
Figure 36. Coding Requirements for Using TCAM Debugging Aids . . . 355
Figure 37. Device Configurations Supported by TCAM (3 parts) . . . 362
Figure 38. Sample JCL for IEDQXA Utility . . . 365
Figure 39. Example of Using the IEBUPDTE Utility (prior to IPL) for Placing a List in SYS1.PARMLIB. . . . 373
Figure 40. TCAM Internal and Device Codes (4 parts) . . . 379
Figure 41. IBM S/360 Internal Code (EBCDIC) . . . 387
Figure 42. USASCII Code . . . 388
Figure 43. Hexadecimal Equivalents for 6-bit Transcode . . . 389
Figure 44. Line Code for IBM 1030 Data Collection System . . . 390
Figure 45. Line Code for IBM 1050 Data Communication System . . . 391
Figure 46. Line Code for IBM 1060 Data Communication System . . . 392
Figure 47. Line Codes for IBM 2260 (Remote)/2265 Display Complexes and IBM 1053 Printer (2 parts) . . . 393

Figure 48. Line Code for IBM 2740 Communication Terminal 395
Figure 49. Hexadecimal Equivalents for IBM 2741 (BCD) Communication Terminal 396
Figure 50. Line Code (EBCD) for IBM 2741 Communication Terminal 397
Figure 51. Line Code (Correspondence) for IBM 2741 Communication Terminal 398
Figure 52. Line Code for AT & T 83B3 and WU 115A Terminals 399
Figure 53. Line Codes for AT & T TWX Terminals 400
Figure 54. Line Code for IBM World Trade Telegraph ITA2 401
Figure 55. Line Code for IBM World Trade Telegraph ZSC3 402
Figure 56. IBM 50 MDI Control Codes 422

Macro Directory

CANCELMG—150
CHECK—269
CHECKPT—151
CKREQ—290
CLOSE
  Application Program—251
  MCP—106
CODE—151
COUNTER—154
CUTOFF—155
DATETIME—155
DCB
  Checkpoint—85
  Input—238
  Line Group—65
  Log—87
  Message Queues—83
  Output—242
ERRORMSG—157
FORWARD—159
GET—264
HOLD—162
ICHNG—282
ICOPY—276
INBUF—146
INEND—147
INHDR—145
INITIATE—163
INMSG—147
INTRO—92
INVLIST—19
LOCK—165
LOCOPT—167
LOG—167
LOGTYPE—44
MCPCLOSE—251
MRELEASE—284
MSGEDIT—168
MSGFORM—179
MSGGEN—180
MSGLIMIT—182
MSGTYPE—183
OPEN
  Application Program—249
  MCP—101
OPTION—26
ORIGIN—185
OUTBUF—148
OUTEND—149
OUTHDR—148
OUTMSG—149
PATH—187
PCB—245
POINT—285
PRIORITY—190
PUT—265
QCOPY—280
QSTART—403
READ—265
READY—104
REDIRECT—192
SCREEN—194
SEQUENCE—196
SETEOF—198
SETSCAN—199
STARTMH—142
TCHNG—281
TCOPY—274
TERMINAL—29
TERRSET—202
TLIST—40
TPEDIT—417
TPROCESS—41
TTABLE—25
UNLOCK—202
WRITE—267

Operator Command Directory

ACTVATED—300
ACTVBOTH—300
AUTOSTOP—301
AUTOSTRT—302
CPRIOPCL—303
DATOPFLD—304
DEBUG—305
DPRIOPCL—307
DSECOPCL—307
ENTERING—308
ERRECORD—308
GOTRACE—310
INACTVTD—311
INTERVAL—311
INTRCEPT—312
LNSTATUS—312
NOENTRNG—313
NOTRACE—314
NOTRAFIC—315
OPTFIELD—316
POLLDLAY—317
QSTATUS—317
RESMXMIT—318
RLNSTATN—319
STARTLINE—319
STATDISP—320
STOPLINE—321
STSTATUS—322
SUSPXMIT—323
SYSCLOSE—324
SYSINTVL—324

How To Use This Book

This book is a reference manual and coding guide for the programmer who must construct or modify a TCAM Message Control Program, or who must write a TCAM-compatible application program. The book assumes familiarity with the overall concepts and structure of TCAM; a good way to achieve this familiarity is to read the *TCAM Concepts and Facilities* publication.

The first seven chapters of the book are concerned with tasks you will encounter in constructing a TCAM Message Control Program (MCP)—such tasks as defining buffers, defining data sets, activating and deactivating an MCP, and actually putting an MCP together. The eighth chapter tells how to make your application programs compatible with a TCAM MCP. Following this is a chapter telling how to use auxiliary services provided by TCAM, such as the checkpoint/restart facility, the operator control capability, and the on-line test function. The final chapter contains information that might be useful in planning and setting up an actual teleprocessing system incorporating TCAM—including TCAM's machine and device requirements, a list of stations supported by TCAM, system-generation considerations unique to TCAM, and directions for preformatting TCAM data sets residing on disk.

Several appendices containing special, helpful information for the system programmer are located in the back of this publication. They include macro instruction formats, transmission code charts, and aids for conversion from QTAM to TCAM. Also of particular interest to the system programmer is the appendix concerning device dependent considerations, which should be read before an MCP is coded. Throughout this publication, wherever a particular device dependency would appear, a reference is made to this appendix instead of listing the individual consideration.

As a first step in familiarizing yourself with this book, look over the table of contents. The book is organized around user tasks, rather than around macros. In defining buffers or terminal- and line-control areas, you must code operands of several macros. If the book were organized around the macros, you would have to look at each operand of each macro to determine which operands pertain to buffer definition, which to terminal- and line-control-area definition, which to incorporating a checkpoint facility, etc. Because the book is organized around tasks, rather than macros, you are saved much of this work. For example, the chapter *Defining Buffers* contains a checklist of those TCAM macro operands having to do with buffer definition. One of the macros mentioned in this checklist happens to be located in the chapter *Activating and Deactivating the Message Control Program,* another is in the chapter *Defining the MCP Data Sets,* a third in the chapter *Defining Terminal and Line Control Areas.* By discussing together those operands having to do with buffers in a section titled *Defining Buffers,* the book saves you the trouble of having to locate these operands yourself when it comes time to design and specify your buffers.

In addition, the task-oriented organization facilitates retrieval of information; to locate information on TCAM's reusable disk queuing scheme you need only relate reusable disk queuing to the task of defining the MCP data sets and look in the table of contents under the chapter-heading *Defining the MCP Data Sets.* Similarly, to locate information on TCAM's checkpoint facility, you need only remember that this is a service facility and look under the chapter-heading *Using TCAM Service Facilities.* Of course, this method of retrieving information by relating it to tasks will work only if you are aware of the tasks we discuss. Each chapter heading shown in the table of contents is the name of one such task.

What the Message Control Program Is

The Message Control Program (MCP) is a set of routines that identify the teleprocessing network to the IBM System/360 operating system, establish line control required for the various kinds of station and modes of connection, and control the handling and routing of messages in accordance with the user's requirements. Every teleprocessing system operated under TCAM requires one MCP.

The MCP serves as an intermediary between the remote stations, and between a remote station and an application program. Device-dependent input/output operations are performed by TCAM routines in the MCP, based on station and line configurations of the system as specified in the operands of TCAM macro instructions in the MCP.

An MCP is coded using a group of TCAM macro instructions. Coding requirements and restrictions for a TCAM macro are identical to those for any other assembler language macro instruction. Assembler language conventions for coding continuations, comments, symbols, and the length, number and format of operands apply to all TCAM macros.

Functions of the MCP

Depending on the requirements of the user, the TCAM MCP might perform any of the following specific functions:

- Enable and disable communication lines.
- Invite terminals to transmit messages.
- Receive messages from terminals.
- Dynamically assign buffers to incoming messages.
- Handle messages on the basis of user-specified priorities.
- Perform message-editing functions for incoming messages. Among such functions are the following: translating from the transmission code to EBCDIC code; deleting line-control characters; inserting time-received and date-received information in the message header; recording the message on a secondary storage medium (logging); inserting or removing user-specified data in the header; maintaining a count of the number of messages received from each station.
- Determine the appropriate destination queue for a message and route the message to that queue.
- Queue the message on the appropriate destination queue.
- Place response messages generated by application programs on queues for subsequent transmission.
- Retrieve messages from destination queues and prepare them for transmission to stations.
- Perform message-editing functions for outgoing messages. Among such functions are the following: placing time-sent and date-sent information in the message header; placing an output sequence number in the header; inserting or removing user-specified data in the header; logging the outgoing message on a secondary storage device; maintaining a count of the number of messages sent to each terminal; inserting line-control characters; translating the message from EBCDIC code to the appropriate transmission code.
- Take periodic checkpoints of the system.
- Provide operator-to-system communications through system control terminals.
- Initiate corrective action when an error or unusual condition is detected.
- Cancel incoming messages containing errors.
- Reroute messages with erroneous header information to a special queue.
- Transmit error messages.

User Tasks in Writing the MCP

As a system programmer concerned with writing a Message Control Program, you will be confronted with five basic tasks:

1. defining the various terminal and line control areas used by the MCP;
2. defining the buffers used by the MCP for handling, queuing and transferring message segments between communication lines and queuing devices;
3. defining the data sets referred to by the MCP;
4. providing for the activation and deactivation of the MCP data sets;

5. defining the Message Handlers, the sets of routines that examine and process control information in message headers, prepare message segments for forwarding to their destination, and route messages to their proper destinations.

In the next five chapters, we shall consider each of these tasks in detail.

In constructing the Message Control Program, the user must provide control information that identifies the remote stations, specifies their characteristics to the system, and tells how they are to be handled by TCAM. This chapter describes how this information is specified.

Terminology

In the following discussion, the word *computer* refers to the central computer in the TCAM system; this is the computer that contains the Message Control Program. Remote terminals, as well as remote computers, are referred to as *stations.*

A *nonswitched line* (also known as a leased or dedicated line) is one over which connections between the computer and remote stations are continuously established. A *switched line* (also known as a dial line) is one over which a direct physical connection between computer and remote station must be established by dialing for data transmission to occur.

A *point-to-point line* connects a single remote station to the computer. Switched lines are considered to be point-to-point. A *multipoint* line connects two or more stations to the computer. For lines to Binary Synchronous (BSC) stations, a line to one station is considered to be multipoint if multipoint BSC data-link control is used on the line.

A *contention line* is one over which the computer and a station may vie for use of the line. Either the computer or a station may "seize" the line, thereby preventing its use by another device on the line until after the device that gained control of the line has transmitted its messages and relinquished control. All TCAM-supported stations not assigned polling or addressing characters, except BSC dial lines, are considered to be contention stations. A *non-contention line* is one for which the computer, using certain user-specified information, determines which station is permitted to enter or accept messages at any particular time.

The computer *sends* a message to a station and *receives* a message from a station; sending and receiving are functions of the computer.

A station *enters* a message to be transmitted to the computer and *accepts* a message transmitted to it from the computer; entering and accepting are functions of a station.

Line Control

Just as a computing system, with its variety of peripheral input/output equipment, requires some means to coordinate the functioning of the various parts, the variety of I/O equipment comprising a teleprocessing system requires a discipline to effectively manage the flow of message traffic. A significant difference should be noted, however. In a conventional computing system, the various I/O devices are at the service of the programmer; the requirements of his program and the characteristics of the data to be processed largely determine which input and output devices are to be activated and when. Moreover, the I/O devices are within reach of the computer operator; he can intervene when a device malfunctions to correct the condition or to assign a different device. In a teleprocessing system, on the other hand, the central computer receives data at random from remote stations, and the operator at the central computer cannot exercise any direct control over remote stations. He cannot, for example, correct a malfunctioning device at a remote station.

A further distinction between a computing system and a teleprocessing system lies in the handling of errors in data. With current techniques for transmitting data over long distances, errors can be introduced into message data by unavoidable transient line conditions such as crosstalk and lightning strikes. Transmission errors occur much less often in a computing system. A discipline for a teleprocessing system must accommodate the facility to detect transmission errors and, when possible, to correct them (as by retransmitting the message containing the errors). If the error cannot be recovered from, its occurrence must be signaled to the user program so that appropriate action can be taken.

The scheme of operating procedures and signals by which a teleprocessing system is controlled is called *line control* (for binary synchronous communications, the term data link control is often used). A line control scheme must consider the functional characteristics and capabilities of the equipment and communication lines comprising the system,

as well as the operational requirements of the system. Some specific factors that line control must consider are: How is contact to be established between a sending and a receiving station? How is a message to be directed to a specific station on a multistation line? What if two stations try to send at the same time? What should be done if a station fails to respond to a message?

Line control can be classified in two ways. The first way is by the transmission technique (start-stop or binary synchronous) used for the line under consideration. With each of these techniques is associated a set of control characters and rules for their use to effect the needed functions. Some of the control characters are used for both start-stop and BSC transmission, while others are peculiar to one or the other of the transmission techniques. For a discussion of these transmission techniques, see the *TCAM Concepts and Facilities* publication.

The second way in which line control can be classified is by the communication line configuration with which it is used. For example, line control for a switched line differs from that for a nonswitched line in the way in which initial contact is made.

While the general capabilities and functions of a given line control scheme are identified in terms of transmission technique and line configuration, individual variations in capability and function arise from differences in the kind of stations to be controlled, and by the presence or absence in the stations of certain features. For example, a given line control scheme may include the control characters needed to indicate occurrence of a transmission error and to request automatic retransmission, but some types of station equipment using that line control scheme may not be capable of error checking or automatic retransmission. Generally speaking, all stations connected to a given line must be designed to use the same line control scheme, and where a certain capability is provided by some stations but not by others, the capability cannot be used.

It is not necessary for the TCAM programmer to specify the line control scheme to be used for a given line; this information is provided implicitly at system generation time, and at assembly time in the DCB macro instruction for the line group of which the given line is a member, and in the TERMINAL macro instructions for the stations on the line. The programmer must, however, have a general understanding of line control concepts to correctly structure that portion of his program involved in message transmission, and to decide intelligently how to deal with line-control characters in his message.

For start-stop stations, the line-control characters recognized by TCAM are EOA and EOB. For BSC stations, the line-control characters recognized by TCAM are STX, ETB and ETX. TCAM removes all of these line-control characters except the EOT from incoming messages if the LC= operand of the STARTMH macro is coded LC=OUT, and leaves them in incoming messages if the LC= operand is coded LC=IN (except that line-control characters are always removed from incoming messages in transparent mode). TCAM inserts line-control characters into outgoing messages if the MSGFORM macro is coded in the outheader subgroup of the Message Handler handling the outgoing message.

If the station that enters the message and the stations that are to accept it are either all similar start-stop or all BSC, and if the user does not wish to change the size of physical blocks of data in the message (if the message is divided into such blocks by EOB or ETB line-control characters), then line-control characters may be left in the message. If the originating and destination stations use different line codes, then the CODE macro must be issued at appropriate places in the MH so that TCAM can translate the message from the line code for the originating station to EBCDIC, then to the line code for the destination station. TCAM's translation tables are set up so that line-control characters for an originating station using one line code are translated into satisfactory characters in the line code for the destination station, provided that the originating station and the destination station are either both similar start-stop or both BSC.

Line control may be left in a message that is processed by a TCAM application program; of course, the user code in the application program will have to take account of line-control characters if they are left in the message.

For a message sent between a start-stop and a BSC station, whether directly or via an application program, the conversion of line-control characters by TCAM's translation tables is less likely to be satisfactory. Figure 40 in *Appendix D* is a chart showing the line-code equivalents of EBCDIC graphic and control characters for each station sup-

ported by TCAM. This chart may be used to determine the character to which TCAM's translation tables will translate an incoming character. For example, an incoming ETB character from a BSC station using EBCDIC line code, if left in the message, will be translated to an EOB character if TCAM's 1050 translation table is used to translate the message from EBCDIC to 1050 line code. (The translation table to be utilized by TCAM is specified by means of the TRANS= operand of the line group DCB macro, while the CODE macro causes translation to be performed and may be used to override the translation table specified in the DCB.)

If the user switching messages between stations having different line codes is satisfied with the equivalent characters provided by TCAM's translation tables, and if he is satisfied with the size of the physical blocks (if any) in his message, he may leave line-control characters in his message; otherwise, he should remove line-control characters from the incoming message by specifying LC=OUT in his STARTMH macro, and insert appropriate line-control characters in his outgoing message by coding a MSGFORM macro in the outgoing group of the Message Handler handling the message.

Operands of MSGFORM permit the user to specify fixed outgoing blocking factors, some of which may be overridden on a terminal-by-terminal basis. The user who wishes to specify physical blocks of data that differ in length within the same message may do so by inserting the appropriate line-control characters in his outgoing message by means of the MSGEDIT macro.

TCAM does not consider the BSC ITB control character to be a line-control character, and does not remove it from incoming messages when LC=OUT is coded in the STARTMH macro. However, the MSGEDIT macro may be used to remove and insert ITB characters, and the BLOCK= operand of the MSGFORM macro may be used to specify a fixed interval at which ITB characters are to be inserted by TCAM into outgoing messages.

For binary synchronous (BSC) stations, another transmission variable involves the treatment of line-control characters in a message. BSC messages may be transmitted in transparent mode or in nontransparent mode.

The *transparent mode* is a type of BSC transmission in which message segments may include certain normally restricted data-link control characters, which are transmitted as ordinary data and not as effective control characters; the only effective data-link control characters transmitted when a message is in transparent mode are those preceded by a DLE data-link character. Transparent mode is useful in transmitting messages containing binary data, fixed- and floating-point data, packed decimal digits, source programs, and object programs, because with such messages the binary structure of a character may be the same as that for a data-link control character.

When a message in transparent mode arrives at the computer, TCAM automatically removes the two initial line-control characters and all effective ETB and ETX control characters. All DLE STX sequences are also removed, except those immediately following an ITB. These characters are removed whether or not LC=OUT is coded in the STARTMH macro. If the user wishes to place a message in transparent mode before sending it to a BSC station, he issues a MSGFORM macro specifying SENDTRP=YES in the outheader subgroup handling messages for that station.

In *nontransparent mode,* all line-control characters are treated as such, and line control is handled as it is for start-stop stations.

In deciding whether to remove and insert line-control characters, and whether messages to BSC stations are to be in transparent mode, the TCAM programmer is concerned with line control at the character level. On a more general basis, he must make decisions regarding those line-control functions used by TCAM to establish contact between the computer and remote stations, and those functions used to maintain an orderly flow of message traffic. The rest of this chapter contains information that will help him to make and to implement these decisions.

Establishing Contact

With TCAM, contact for the purpose of message transmission may be established in several ways, depending upon the line configuration and the stations involved. Contact is always established under the control of the central computer, which performs (in the channel) a number of set-up or preparatory operations, which are followed by either a Read or a Write operation on the line (except when the set-up operations determine that

the remote station is not free to enter or accept data, in which case no message transmission occurs).

In this publication, when contact is established for the purpose of receiving data from a station, the process is called *invitation*; when contact precedes the sending of data to a station, the process of establishing contact is called *selection.* Selection is performed when the central computer has a message to send to a station; invitation is performed to give a station the opportunity to enter a message if it has one ready (in some cases of invitation, the station initiates contact with the computer to enter a message and the computer completes the invitation process).

Invitation

There are two forms of invitation, that involving *contention* (with or without identification sequence exchange) and that involving *polling.*

In a TCAM system either the computer or a station on a point-to-point contention line can bid for use of the line so that it can send a message to another device. In some configurations, it is possible for both computer and station to simultaneously bid for the line; when this happens, the computer and station are said to contend with each other (hence the name *contention line*).

For contention-type stations, *invitation* by TCAM means that TCAM gives the station an opportunity to enter data, that is, TCAM "listens" on the line for a signal from the station indicating that the station wishes to enter a message.

The alternative to the contention form of invitation involves a system in which the central computer periodically examines each active entry in an invitation list (discussed below) of remote stations, and invites each station to enter any input messages it has ready. Each station in the list has a unique identifier, usually consisting of one or two characters which cause that station, and no other, to respond. The process of contacting each remote station in this manner is called *polling*, and the station identifiers are called *polling characters.* Often, the first polling character identifies the station and the second identifies a particular component of the station.

For polled nonswitched lines, TCAM commences invitation by polling the first station listed in the invitation list for the line. (If the line is point-to-point, the first will be the only station on it, and the invitation list for the line need contain only one entry.) TCAM uses polling characters unique to each station to ask each if it has a message to send. If the response is negative, or if there is no response (i.e., if the station is down), the polling characters for the next station listed are sent; this process is repeated until a station responds positively by entering a message. Such a station is permitted to enter any messages it may have ready for the computer and may be sent any messages that are queued for it (see *Transmission Priority* in this chapter for a discussion of when sending to a station may occur relative to receiving). After all messages are entered, the computer interrogates the next station in the invitation list. After all the active stations in an invitation list for a given polled line have been invited to enter a message, a delay equal to the number of seconds specified in the INTVL= operand of the DCB macro for the line group may be observed to allow for sending before polling is restarted at the beginning of the list; if this operand is omitted, no delay occurs. The polling interval reduces unproductive polling on lines that are not used continually.

For nonswitched polled lines, the computer initiates contact with the stations. However, for switched lines the station may initiate the contact by successfully dialing the computer. The polling function in this case consists only of sending the polling characters to the station that initiates the contact. The station responds by entering one or more messages. The computer sends the polling characters after each message is received.

It is possible for the computer to dial some types of polled stations on switched lines. The user may specify computer-initiated contact by coding certain operands of the TERMINAL macro, discussed below. In this case, the polling characters for the station are sent once contact has been established and all messages queued for the station have been transmitted.

Constructing the Invitation List

A TCAM system maintains control of invitation by means of an invitation list for each line. An invitation list is created by means of an INVLIST macro instruction.

INVLIST Macro Instruction

The INVLIST macro

- generates the invitation list for a line;
- specifies active and inactive invitation list entries;
- is required for each line in the system (though the same INVLIST macro may be sufficient for more than one output-only line);
- is specified following the macros defining the terminal table.

One INVLIST macro must be issued for each line in the system, with the exception of output-only lines to stations that do not use invitation sequences; a single INVLIST macro is sufficient for all such output-only lines. The names of all INVLIST macros for the lines in a line group must be specified, by ascending relative line number, in the INVLIST= operand of the DCB macro for the line group.

For each station on a line, the INVLIST macro creates an invitation list entry that contains the invitation characters for the station (the polling characters for polled stations, or the identification sequence assigned to TWX, World Trade, and switched BSC stations using such a sequence). See *Appendix G. Device Dependent Considerations,* for particular invitation list specifications for the:

- 2260 Display Stations, both local and remote;
- 2740 Communications Terminal with the Station Control or Station Control and Checking feature;
- BSC terminals;
- 2740 Communications Terminal with the Transmit Control or Transmit Control and Checking feature;
- TWX terminals;
- 7770 Audio Response Unit, Model 3.

No invitation characters are present in entries for contention terminals not assigned identification sequences.

The INVLIST macro has the following format:

Name	*Operation*	*Operand*
symbol	INVLIST	ORDER= (entry,...)[,EOT=hexchars] [,CPUID=addr]

symbol

Function: Specifies the name of the macro and of the invitation list for the line.
Default: None. This name must be specified.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).
Notes: This name must be the same as a name specified by the INVLIST= operand of the DCB macro for the line group containing this line.

ORDER= (entry,...)

Function: Specifies the invitation list entries for the line.
Default: None. For all output-only lines to stations having no ID sequence assigned to them, specification optional. For all other cases, this operand must be specified.
Format: The exact manner in which each entry is coded is described below. A maximum of 200 entries may be coded.
Notes: For polled lines, there must be at least one entry for each station that can enter messages on the line. Entries are specified in the order in which the stations are to be invited to send messages.

EOT= hexchars

Function: Specifies the EOT line-control character for the stations on this line.
Default: None. For lines to multipoint BSC stations, this operand must be specified. For all other cases, this operand must not be specified.
Format: A single hexadecimal character, unframed, in the transmission-code representation.
Notes: Appropriate EOT characters are as follows:

- for EBCDIC: 37
- for ASCII: 04
- for 6-bit Transcode: 1E

Example:
For a line to multipoint BSC stations using ASCII as their transmission code, this operand would be coded as follows:

EOT=04

where 04 is the ASCII transmission code representation of the EOT control character, in hexadecimal notation.

CPUID=addr

Function: Specifies the name of a field containing the ID sequence assigned to the computer.
Default: None. For switched lines to stations using ID sequences, when the computer is expected to exchange ID sequences with stations on the line when calls are made, this operand must be specified. For all other cases, specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: The field named by *addr* should consist of a length byte, specifying in binary form the number of characters in the computer ID sequence, followed by the ID sequence itself in line code. For more information on ID sequences, see *Switched Lines to Stations Using ID Sequences* below. This operand also specifies the invitation message for adio terminals.

Example:
For a switched line to stations using ID sequences and EBCDIC line code, this operand might be coded as follows:

CPUID=CPUNAME

Somewhere within the same area of addressability in the MCP the following field might be defined:

```
CPUNAME  DC X '04'
         DC X 'D5D6D3C1'
```

Here, X'04' is the hexadecimal number of bytes in the rest of the field, while X'D5D6D3C1' is the EBCDIC character sequence NOLA in hexadecimal notation.

Each entry specified as a suboperand of the ORDER= operand consists of a station or line name, an indicator that determines whether the station represented by the entry is initially capable of entering messages, and a sequence of invitation characters.

The station name must be the name of the TERMINAL macro for the station being entered in the list.

The indicators to distinguish active from inactive entries are as follows:

+ indicates that the station represented by the entry is initially activated for entering messages.

– indicates that the station represented by the entry is not initially activated for entering messages.

Entries may be activated or deactivated for entering, accepting or both, by means of various operator commands or by an ICHNG macro issued in an application program. When polling is used as the method of invitation, only stations that are activated for entering are polled.

Following the indicator in an entry are the invitation characters for the station. These will be either polling characters or an identification sequence. Invitation characters are generally assigned to a station when it is installed. For information on whether a particular station can be assigned identification or polling characters, consult the hardware manual for that station. Invitation characters are specified in transmission-code representations, converted to hexadecimal notation. (For conversion tables, see *Appendix G*). Each group of invitation characters in a list must be of the same length.

An invitation list entry might be coded as follows:

NYC+E40D

Here NYC is the name of an IBM 1050 terminal in New York City, + indicates that this entry is active for entering messages, and E40D is the IBM 1050 transmission code representation of the polling characters B6 in hexadecimal notation.

NOTE: Because the operand field of a macro is limited to 255 characters, TCAM provides a facility to specify additional INVLIST entries if necessary. A comma placed as the last character of the last entry field; i.e.,

ORDER= (entry,entry,...entry,)

indicates a continuation of the macro. The next source statement would then be coded

INVLIST ORDER= (entry,entry,...)

There is no limit (other than the maximum of 200 entries that may be specified) on the number of continuation statements used.

The exact manner in which the INVLIST macro is coded depends upon the line configuration and upon station features. The following paragraphs describe the possible ways in which INVLIST may be coded.

Nonswitched point-to-point or multipoint lines to stations using polling characters

Issue one INVLIST macro for each such line, and code at least one entry for each station (active and inactive) on the line. Each entry should include the terminal name, the active/inactive-entry indicator, and the polling characters assigned to the terminal. If a terminal is to be polled more than once in one pass through the invitation list, specify more than one entry for this terminal—the terminal will be polled once for each active entry specified. To poll a specific component of a terminal, specify the second polling character, which identifies that component.

Example 1:
The following INVLIST macro creates the required invitation list for a nonswitched multipoint line having three IBM 1050s as terminals.

Name	*Operation*	*Operands*
LIST1	INVLIST	ORDER= (NYC+E40D,BOS+E20D,NYC+E40D, PHI–E715)

TCAM uses the invitation list created by this macro to poll the IBM 1050 terminals located in New York City (NYC), Boston (BOS), and (again) New York City, in that order. The New York City terminal is polled twice as often as the Boston terminal. The Philadelphia terminal (PHI) is inactive until activated by the operator control facility or by an ICHNG macro issued in an application program. E40D, E20D, and E715 are the IBM 1050 transmission code representations of the polling characters B6, A6, and C0, respectively, in hexadecimal notation. + means the terminal is initially active; – means the terminal is initially inactive.

Example 2:
The following INVLIST macro creates the invitation list for a nonswitched multipoint line having one BSC IBM 2780 and one BSC IBM 1130, using the Auto Poll hardware feature.

Name	*Operation*	*Operands*
LIST2	INVLIST	ORDER= (BAL+C2F62D,DET+32C42D),EOT=37

TCAM uses the invitation list created by this macro to autopoll an IBM 2780 located in Baltimore (BAL) and an IBM 1130 located in Detroit (DET), in that order. The transmission code for the terminals being autopolled is EBCDIC; the C2F6 and C4 are the hexadecimal-notation form of the EBCDIC representation of the polling characters B6 and D, respectively. The 2D ending each entry is the hexadecimal-notation form of the EBCDIC representation of the ENQ line-control character, which must be included with all BSC polling sequences. The "32" in the Detroit entry is the hexadecimal-notation form of the EBCDIC SYN character, used to pad the DET polling sequence to the length

of the BAL sequence. The EOT= operand presents the hexadecimal form of the EBCDIC EOT character; it must follow all entries in an INVLIST macro for autopolled terminals using EBCDIC transmission code.

Switched lines to terminals using polling characters

Issue one INVLIST macro for each such line. Polling characters for all polled terminals assigned to a switched line (by means of each terminal's TERMINAL macro, described below) must be identical. Since all terminals assigned to the same line have the same polling characters, it is necessary to code only one representative entry as the operand of the INVLIST macro for a line; this entry names any one of the terminals assigned to the line, and gives the polling characters for all terminals assigned to the line. If a TERMINAL macro with the operand UTERM=YES is issued for the line, code the name of that TERMINAL macro, rather than the name of a terminal, in the representative entry.

Example:

The following INVLIST macro creates the invitation list for a switched line having three polled IBM 1050 terminals (NYC, BOS, and PHI) assigned to it.

Name	*Operation*	*Operand*
LIST3	INVLIST	ORDER=(NYC+E40D)

Whenever one of the three terminals calls in (or is called), TCAM uses the polling characters represented in hexadecimal notation by E40D to invite it to enter a message. E40D is the IBM 1050 transmission code representation of the polling characters B6, in hexadecimal notation.

Note that only one of the three terminals is used to create the entry in the invitation list. If this entry were inactive (i.e., if – rather than + were coded), none of the three terminals assigned to the line could enter messages.

Switched lines to stations using ID sequences

Issue one INVLIST macro for each such line. Code one entry for each ID sequence assigned to one or more stations on the line, and code the CPUID= operand if the computer is assigned an ID sequence. Each ID sequence is entered in its transmission code representation, converted to hexadecimal notation. No framing characters or quotes are used.

If each station assigned to a switched line has its own unique ID sequence, then one entry is coded for each station. Each entry consists of the station name, the active/inactive-entry indicator, and the ID sequence assigned to the station. (See Example 1 below.)

If two or more stations assigned to a switched line share the same ID sequence, then one entry is coded for each different ID sequence assigned to a station or stations on the line. If a TERMINAL macro specifying UTERM=YES is issued for the line, then each entry consists of the name of the TERMINAL macro, the active/inactive-entry indicator, and an ID sequence. If no such TERMINAL macro is issued, then each entry consists of the name of a representative station using the ID sequence mentioned in this entry, the active/inactive-entry indicator, and an ID sequence. (See Example 2 below. For guidance on when to code a TERMINAL macro using UTERM=YES, see the discussion of the TERMINAL macro.)

NOTE: If a switched station calls in and enters an ID sequence, TCAM uses the ID sequence to establish the origin of messages entered by the station. If a switched station assigned a non-unique ID sequence and represented by an invitation-list entry specifying the name of a TERMINAL macro coded for a line calls in and fails to identify itself by means of an origin field in a message header, the station will not receive any messages during the call (because TCAM does not know whose messages to send unless there are messages queued for the line). If a switched station assigned a non-unique ID sequence and represented by an invitation-list entry specifying the name of a representative station using the ID sequence calls in and fails to identify itself by means of an origin field in a message header, during the call the station receives those messages queued for the station named in the invitation-list entry, even if the calling station and the station named in the invitation-list entry are two different stations.

If a switched station calls in and enters an ID sequence, TCAM searches for the ID sequence in the invitation list associated with the line on which the station called in. If the ID sequence is not found in an entry in the invitation list for this line, TCAM conducts a search of the invitation lists for any lines in this line group which have a higher relative line number than that assigned to the line over which the station called in. TCAM searches these invitation lists according to ascending relative line number until either the ID sequence is found in a list or the invitation list for the highest-numbered line in the line group has been searched. If the ID sequence is found, TCAM assumes that the station associated with that ID sequence in the invitation list is the calling station, and maintains the connection. If the ID sequence is not found, TCAM breaks the connection with the calling station, thereby freeing the line.

Example 1:
The following INVLIST macro creates the invitation list for a switched line having three IBM 2770 terminals (named NYC, BOS, PHI) assigned to it. Each of these terminals is assigned a unique ID sequence; that for NYC is AA, that for BOS is BB, while that for PHI is CC. PHI is not to be initially eligible for entering data. The computer is assigned the ID sequence POK1. The stations use EBCDIC line code.

Name	*Operation*	*Operands*
LIST4	INVLIST	ORDER=(NYC+C1C1,BOS+C2C2,PHI–C3C3), CPUID=IDFIELD

Here C1C1, C2C2, and C3C3 are the EBCDIC transmission-code representations of the ID sequences AA, BB, and CC, respectively, in hexadecimal notation. Somewhere within the same area of addressability in the MCP the following field is defined:

Name	*Operation*	*Operand*
IDFIELD	DC DC	X'04' X'D7D6D2F1'

Here, 04 is the hexadecimal length of the rest of the field. D7D6D2F1 is the EBCDIC representation of the ID sequence POK1, in hexadecimal notation.

Example 2:
The following INVLIST macro creates the invitation list for a switched line having six IBM 1130 stations assigned to it. Three of these stations are assigned the ID sequence BATCH1, while the remaining three are assigned the ID sequence BATCH2. The computer is assigned the ID sequence RAL. The stations used EBCDIC transmission code. A TERMINAL macro with UTERM=YES specified, named RELLN3, has been issued for this line.

Name	*Operation*	*Operands*
LIST5	INVLIST	ORDER=(RELLN3+C2C1E3C3C8F1, RELLN3+C2C1E3C3C8F2), CPUID=IDADDR

Here, C2C1E3C3C8F1 and C2C1E3C3C8F2 are the EBCDIC transmission-code representations of the ID sequences BATCH1 and BATCH2, respectively, in hexadecimal notation. Somewhere in the MCP the following field is defined:

Name	*Operation*	*Operand*
IDADDR	DC DC	X'03' X'D9C1D3'

Here, 03 is the hexadecimal length of the rest of the field. D9C1D3 is the EBCDIC transmission-code representation of the ID sequence RAL, in hexadecimal notation.

Switched or nonswitched contention lines to terminals not assigned ID sequences

If only one station is assigned to the line, include in the entry portion of the macro the station name and the active/inactive indicator. If more than one station is assigned to the line, include in the entry portion of the macro the name of a representative station and

the active/inactive indicator.

For IBM 2740 Basic and IBM 2780 stations for which equal priority is specified in the line group DCB macro, include in the entry portion of the macro the station name, the active/inactive indicator, and, optionally, a single dummy polling character in hexadecimal notation (any character will do).

Example:
The following INVLIST macro creates the invitation list for a nonswitched line to an IBM 2740 Basic terminal in New York City (NYC).

Name	*Operation*	*Operand*
LIST6	INVLIST	ORDER= (NYC+)

Output-only lines to stations having no ID sequences assigned to them.
Issue one INVLIST macro to serve all such lines; the name of this macro should be specified in the INVLIST= operand of the DCB macro for each output-only line group. No operand is coded for this INVLIST macro. (Stations having ID sequences assigned to them must appear as entries in the INVLIST macro for their line, regardless of whether or not the line is output-only.)

Example:
The following INVLIST macro creates the invitation list for all output-only lines to stations having no ID sequences assigned to them.

Name	*Operation*	*Operand*
LIST7	INVLIST	

Selection

Selection is the process by which contact is established between the computer and a station for the purpose of transmitting data from the computer to the station.

As is the case with invitation, there are basically two forms of selection. One of these is used with contention stations (which may or may not be equipped with a feature permitting identification sequence exchange), while the other involves transmission by the computer of addressing characters (similar to polling characters) to a station preparatory to sending the station a message. Response to the transmission of these characters indicates whether the terminal can accept the message.

The contention form of selection is similar to the contention form of invitation, described above. When the computer has a message to send to a contention station, it waits until the line is free of traffic and then seizes it; once it has control of the line, the computer merely sends the message to the terminal.

When addressing characters are used in selection, the selection process is closely related to the polling form of invitation, described above. That is, the flow of messages to and from a station is controlled by the computer according to an orderly scheme. The nature of this scheme is discussed in the section *Maintaining Orderly Message Flow.* Addressing characters are defined in the TERMINAL macro.

Constructing the Terminal Table

In selecting a station or application program, TCAM uses information provided by TCAM macros at assembly time and stored in control areas. The control areas used in selection all depend upon the terminal table. The *terminal table* consists of blocks of information about each station and application program; each such block is called a *terminal entry.*

There are eight types of terminal entry:

- A *single entry* in the terminal table defines a single station. A single entry is created by means of a TERMINAL macro; one such entry must be created for each station in the system that is not defined by a group or line entry (see below).
- A *group entry* represents a group of terminals on a line that have the Group Code hardware feature; specification of a single set of unique addressing characters results in simultaneous transmission of a message to all terminals in the group. If a terminal that is a member of a group is also to be addressed individually, or is to be polled, it must

also be represented by a single entry. A group entry is defined by a TERMINAL macro and is for output only.

- A *component entry* defines a component of a station that may be individually addressed—for example, a card reader or a printer on an IBM 1050 station. If more than one component of a station may be individually addressed, a component entry may be required for each. A component entry is defined by a TERMINAL macro.
- A *line entry* defines a switched line that is used for input or input/output operations. The line entry is used to supply device characteristics for stations that call in on a switched line before they identify themselves (by the origin field in a message header, as checked by an ORIGIN macro in the Message Handler), and for stations that call in and never identify themselves. The entry is defined by a TERMINAL macro specifying UTERM=YES.
- A *distribution list entry* contains a list of pointers to single, group, cascade or process entries. When a message or FORWARD macro contains the list name as its destination code, TCAM sends the message by separate transmissions to all stations indicated by the list. Each station on the list must have a corresponding single or group entry in the terminal table. A distribution list entry is defined by a TLIST macro.
- A *cascade list* entry contains a list of pointers to single, group, or process entries. When a message or FORWARD macro contains the list name as its destination code, the message is queued to be sent to that single valid station or opened process entry in the list that has the least number of messages queued for it. A valid station is one that is capable of accepting a message, and that is on a line for which the line group data set has been opened. If more than one valid station has the smallest number of messages queued, the message is queued for the first of these. If no station is valid or if all queues are of the same length, the message is queued for the first station in the list. A cascade entry is defined by a TLIST macro.
- A *process entry* represents an application program. One process entry must be defined for each queue to which an application program can issue a GET or READ and at least one must be defined for all PUTs or WRITEs from the same application program. One open input or output DCB in the application program is associated with each process entry. A process entry is defined by a TPROCESS macro.
- A *logtype entry* represents a queue of complete messages for a logging medium. A logtype entry is defined by a LOGTYPE macro.

The size, structure, and contents of the terminal table depend upon information provided by the user through the TTABLE, OPTION, TERMINAL, TLIST, TPROCESS, and LOGTYPE macro instructions. These macros are described in this chapter.

Macro instructions defining the terminal table are coded as a group. For an example of a coding sequence for a terminal table, see the chapter *Putting the MCP Together.*

TTABLE Macro Instruction

The TTABLE macro:

- defines the start of a terminal table,
- names the last entry in the table,
- is required as the first macro defining the terminal table,
- is issued only once.

An operand of TTABLE specifies the name of the last macro issued in the section of code defining the terminal table; thus, TTABLE defines the beginning and end of the terminal table coding section. The TTABLE macro must be followed immediately by the macros defining the terminal table.

The TTABLE macro has the following format:

Name	*Operation*	*Operand*
[symbol]	TTABLE	LAST=name[,MAXLEN=integer]

Function: Specifies the name of the macro and the name of the terminal name table (an internal table associated with the terminal table).
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

LAST=name

Function: Specifies the name of the last entry in the terminal table (i.e., the name of the last TERMINAL, TLIST or TPROCESS macro coded).
Default: None. This operand must be specified.
Format: Must conform to the rules for assembler language symbols.

MAXLEN=integer

Function: Specifies the number of characters in the name of the terminal table.
Default: None. Specification optional.
Format: An unframed decimal integer.
Maximum: 8
Notes: If this operand is omitted, the length of the last entry is assumed. The operand is not necessary if the lengths of all terminal table entry names are the same, or if the last entry has the greatest length.

OPTION Macro Instruction

The OPTION macro:

- Permits space to be reserved for an option field related to a station, component, line, or application program.
- Must be specified prior to any TERMINAL, TLIST, or TPROCESS macros.
- Is optional among the macros defining the terminal table.

OPTION macros are issued as a group; in conjunction with the OPDATA= operands of the TERMINAL and TPROCESS macros they define the *option table,* a storage area containing option fields related to individual stations, components, lines, or application programs. The option fields are accessed by certain Message Handler routines that need source- or destination-related storage in order to perform their functions. Among the MH macros that invoke routines that access the option fields are the following: STARTMH, INHDR, INBUF, INMSG, OUTHDR, OUTBUF, OUTMSG, COUNTER, ERRORMSG, FORWARD, LOCOPT, MSGLIMIT, PATH, and REDIRECT. To gain some insight into the function of option fields, the reader should turn to the individual discussions of these macros in the chapter *Designing a Message Handler.* User-written routines can also access information in an option field.

Taken together, the OPTION macros issued by a user define a complete set of option fields; all or part of this set may be assigned to a particular station, component, line, or application program by means of the OPDATA= operand of the TERMINAL or TPROCESS macro (see the example below). An OPTION macro merely gives an option field a name and describes the type and length of the field in assembler-language format; an area of storage is neither initialized nor actually allocated for the field unless the field is specified for a particular station, component, line, or application program by means of the OPDATA= operand of the TERMINAL or TPROCESS macro. Up to 254 option fields, each of which may be up to 255 bytes long, may be defined in an MCP by OPTION macros. All or any part of the set of option fields may be allocated to each station, component, line, or application program represented by a terminal-table entry. For the set of option fields for a particular entry in the terminal table, the last option field must be within 254 bytes of the first.

A new area of storage having the name and attributes specified by the OPTION macro defining an option field is assigned to each station, component, line, or application program whose TERMINAL or TPROCESS macro initializes that field. Each TERMINAL or TPROCESS macro may initialize a field differently; hence different stations, components, lines, or application programs may be assigned option fields having identical names and attributes, but different contents. This feature allows the user to tailor the functions of a macro accessing an option field to meet the needs of a particular station, component,

line, or application program. For example, the COUNTER macro maintains a count of messages or message segments received from or sent to a station. This counter is located in an option field for that station. If the OPTION macro for this field is named COUNT, and if the COUNTER macro names COUNT as the field in which the counter should be maintained, then a separate counter will be maintained for each station that uses the OPDATA= operand of the TERMINAL macro to initialize COUNT.

A macro coded in an inheader, inbuffer, or inmessage subgroup handling messages entered by stations on a line accesses the specified option field for the station that entered the message being processed. (If the originating station is unknown because it called in on a switched line and failed to identify itself, the specified option field for the line entry associated with this line is accessed.) A macro coded in an outheader, outbuffer, or outmessage subgroup handling messages destined for stations on a line accesses the specified option field for the station that is to accept the message being processed. A macro coded in an outheader, outbuffer, or outmessage subgroup handling messages destined for an application program, accesses the specified option field associated with the process queue to which the GET or READ macro that is moving this message to the application program is directed. A macro coded in an inheader, inbuffer, or inmessage subgroup handling messages being received from an application program accesses the specified option field for the process entry associated with the DCB named in the PUT or WRITE macro.

The OPTION macro has the following format:

Name	*Operation*	*Operand*
opfldname	OPTION	typelength

opfldname

Function: Specifies the name of the option field.
Default: None. This name must be specified.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

typelength

Function: Specifies the type and length of the option field.
Default: None. This operand must be specified.
Format: Standard assembler language format (e.g., H, CL8, AL3). All assembler language type codes may be utilized. However, B, C, P, X and Z must be coded with a length attribute (e.g., CL5, BL4). Duplication factors are not allowed; i.e., ABC OPTION 3DL5 is an invalid macro.
Notes: When the option field is used in conjunction with the FORWARD, ERRORMSG, or REDIRECT macro, a character string of length *n* must be specified, where *n* is the length in bytes of the data in the OPDATA= operand of the TERMINAL or TPROCESS macros that initialize the fields.

If used with counter, *typelength* should be specified as H, since this macro requires a halfword field on a halfword boundary.

If used with INBUF, INHDR, INMSG, OUTBUF, OUTHDR, OUTMSG, PATH, or MSGLIMIT macros, *typelength* should specify a one-byte field (e.g., FL1, AL1). No boundary alignment is required.

If used with STARTMH, *typelength* will specify a one- or four-byte field, depending upon which STARTMH operand names the option field.

Points to remember:

- OPTION macros, if used, must be issued as a group and must immediately follow the TTABLE macro.
- The order in which OPTION macros are arranged determines the order in which initialization data must be specified in the OPDATA= operand of the TERMINAL or TPROCESS macro. If a field specified by an OPTION macro is not to be defined for a particular station, component, line, or application program, then a comma should be coded in place of the data for this field in the OPDATA= operand (but trailing commas should not be coded).

- OPTION macros should be arranged so as to prevent waste of storage space in the option table. For example, if three OPTION macros are coded

```
AA OPTION FL1
AB OPTION CL4
AC OPTION H
```

  the halfword specification for the AC field causes the assembler to perform boundary alignment. Since the AC field may not already be on a halfword boundary, one byte of storage area in the option table may be wasted for each terminal for which these option fields are defined. To conserve storage space, the above macros should be coded as follows:

```
AC OPTION H
AA OPTION FL1
AB OPTION CL4
```

  if four OPTION macros are coded

```
BA OPTION F
BB OPTION CL1
BC OPTION H
BD OPTION CL1
```

  two bytes of storage area in the option table will be wasted for each station after the first for which these option fields are defined. To conserve storage space, the above macros should be coded as follows:

```
BA OPTION F
BC OPTION H
BB OPTION CL1
BD OPTION CL1
```

- In coding an OPTION macro, the user must specify the type and length of the option field to be generated. This information is contained in the discussion of the individual macro that accesses the option field.

Example:
In the following example, the TTABLE macro defines the begining and end of the terminal table section of the Message Control Program. The OPTION macros, which are a part of this section of code, define fields in the option table that are accessed by the COUNTER, MSGLIMIT, REDIRECT, ERRORMSG, and PATH macros.

Name	*Operation*	*Operand*
	TTABLE	LAST=PROC
COUNT	OPTION	H
MSGLMT	OPTION	FL1
REDRECT	OPTION	CL3
ERRMSG	OPTION	CL4
PATHSW	OPTION	FL1

TTABLE defines PROC as the name of the last entry in the terminal table. The OPTION macros define an 11-byte optional area for entries in the terminal table. The optional area consists of five fields:

- COUNT defines a halfword for decimal data to be used by the COUNTER macro.
- MSGLMT defines one byte for decimal data to be used by the MSGLIMIT macro.
- REDRECT defines a character string consisting of three bytes naming the terminal; this data is used by the REDIRECT macro.
- ERRMSG defines a character string consisting of a four-byte terminal name; this data is used by the ERRORMSG macro.
- PATHSW defines one byte for eight binary path switches to be tested by various delimiter macros.

If the OPDATA= operand of a TERMINAL macro were coded

```
OPDATA=(0,0,NYC,PITT,3)
```

an 11-byte storage area would be set aside in the option table for use by MH macros in handling messages to and from that terminal. The COUNT and MSGLMT fields would initially contain 0, the REDRECT field would contain NYC, the ERRMSG field would contain PITT, and the PATHSW field would contain 3.

If the OPDATA= operand of another TERMINAL macro were coded

```
OPDATA=(,,NYC,PITT)
```

a 7-byte storage area would be set aside in the option table for use by MH macros in handling messages to and from that terminal. Only the REDRECT and ERRMSG fields would be created.

Note that for an option field to be created for any particular terminal, two conditions must be satisfied:

1. An OPTION macro defining the field must be issued.
2. The field must be initialized in the OPDATA= operand of the TERMINAL macro for that terminal. If a comma is coded in place of a field in the OPDATA= operand, no space is set aside for that field. If the OPDATA= operand of a TERMINAL macro is omitted, no option fields are set aside for that terminal.

TERMINAL Macro Instruction

The TERMINAL macro:

- Creates a single, group, or line entry in the terminal table.
- Specifies the type of queuing to be used (i.e., queuing by line or queuing by terminal),
- Specifies the addressing characters to be used in addressing a station,
- Specifies when the computer is to initiate contact with switched stations,
- Specifies how often the computer is to initiate contact with switched stations,
- Designates secondary operator control stations,
- Specifies initial data for the option table,
- Specifies an alternate destination for messages sent to the station for which this TERMINAL macro is issued,
- Overrides the buffer size specified by the BUFSIZE= operand of the line group DCB, for output only,
- Specifies blocking factors to be used for inserting control characters in outgoing messages destined for this station, when a MSGFORM macro is executed in an out-header subgroup handling such messages,

  Is required for each single or group station or line entry in the TCAM system.

The TERMINAL macro causes an EBCDIC station or line name, and information associated with the station or line, to be included as an entry in the terminal table. If a single station or component is involved, TERMINAL produces a single entry in the terminal table. If a group of stations having the group-code feature is involved, TERMINAL produces a group entry. If a line is involved, TERMINAL produces a line entry.

One TERMINAL macro should be coded for:

1. Each station (whether switched or nonswitched) that can accept messages, and for some terminals that can only enter messages (see *Coding the TERMINAL Macro for a Line* below).
2. Each group of nonswitched terminals equipped with the hardware group-code feature. Terminals can only accept messages under the group-code feature; they cannot enter messages. Each terminal in the group that can also enter messages, or that can be addressed separately, must also be represented by a single entry.
3. Each switched line to stations that do not uniquely identify themselves after calling the computer.

For guidelines on coding the TERMINAL macro for a line and for a component, see the next two sections of this chapter.

TERMINAL macros for stations on a line must be issued together, and the groups of TERMINAL macros for each line in a line group must be contiguous and in ascending relative line sequence.

When TERMINAL macros are issued for the individual components of a station, the macros for the components must immediately follow that for the station.

NOTE: See *Appendix G. Device Dependent Considerations,* for particular specifications for the

- 1030 Station;
- 2260 Display Station (remote);
- 2740 with Station Control or Station Control and Checking feature;
- 2740 with the Transmit Control or Transmit Control and Checking feature;
- 2740 Basic Terminal;
- 2740 Model 2 Communications Terminal;
- 2770 Data Communications System;
- BSC stations;
- AT&T 83B3 stations.

The TERMINAL macro has the following format:

Name	*Operation*	*Operand*
symbol	TERMINAL	QBY={T / L} ,DCB=dcbname ,RLN=integer ,TERM=type ,QUEUES=form [,DIALNO={chars / NONE}] [,ADDR=chars] [,LEVEL=(integer,...)] [,CLOCK=time] [,CINTVL=integer] [,BUFSIZE=integer] [,ALTDEST=entry] [,BFDELAY=integer] [,NTBLKSZ=(blocksize,subblocksize)] [,TBLKSZ=integer] [,OPDATA=(data,...)] [,SECTERM={YES / NO}] [,COMP={YES / NO}] [,UTERM={YES / NO}]

symbol

Function: Specifies the station name.
Default: None. This name must be specified.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).
Notes: This name can appear in an origin or destination field of a message header. SYSCON may not be used.

If more than one TERMINAL macro is specified for the same buffered station (by coding two TERMINAL macros, with different names, for the same station), message segments may become intermixed during sending or receiving operations. Furthermore, a text segment may be treated as a header segment. For these reasons, coding more than one TERMINAL macro for the same buffered station is not a recommended procedure, unless the TERMINAL macros are coded for components rather than for the station itself.

QBY={**T**|**L**}

Function: Specifies the type of message queuing.
Default: None. This operand must be specified.
Format: T or L.
Notes: T specifies that outgoing messages are to be queued by station; that is, all messages for a given station on a line are sent in priority order before any messages for other stations on that line are sent (except for 2770 stations for which BFDELAY= is coded; messages are sent to these stations a buffer at a time). T should be specified for switched stations, and must be specified for stations using TCAM's buffered-terminal support. For a more complete discussion of queuing by station, see *Maintaining Orderly Message Flow* in this chapter.

L specifies that outgoing messages are to be queued by line; messages for all stations on the line are sent on a first-ended first-out basis within priority groups. If L is specified for stations on a switched line, when contact is made with a station on that line all messages on the queue are sent to that station, regardless of what station they are intended for. For a more complete discussion of queuing by line, see *Maintaining Orderly Message Flow* in this chapter.

This operand is ignored if the TERMINAL macro is coded for a component or for a line.

DCB=dcbname

Function: Specifies the name of the data control block for the line group in which the station is included.
Default: None. This operand must be specified.
Format: Must conform to the rules for assembler language symbols.

RLN=integer

Function: Specifies the relative line number, within the line group, of the access line over which the computer and the station communicate.
Default: None. This operand must be specified.
Format: An unframed decimal integer.
Maximum: 255.
Notes: For a discussion of how relative line numbers are assigned, see *DD Statements for a Line Group.* For a switched station on a line for which no TERMINAL macro coded for a line is issued, any access line in the group may be specified. When the computer calls a station assigned to a switched line, it attempts to make the call using the line whose relative line number is specified. If that line is unavailable, the line whose relative line number is greater than that specified by *integer* is examined; this process is repeated until a free line is found or until all lines in the group that have relative line numbers higher than the *integer* specified for this station have been examined. If all higher-numbered lines in the line group are unavailable, the station is not dialed at this time. Dialing is postponed until a suitable line is available.

TERM=type

Function: Specifies the terminal type.
Default: None. This operand must be specified.
Format: This operand may be replaced by any of the following values: 1030, 1050, 1060, 226L (2260 Local), 226R (2260 Remote), 2265, 274A (nonswitched Basic 2740 Model 1), 274B (switched 2740 Model 1), 274C (nonswitched 2740 Model 1 with Station Control), 274D (nonswitched 2740 Model 1 with Station Control and Checking), 274E (switched 2740 Model 1 with Transmit Control and Checking), 274F (nonswitched 2740 Model 1 with Checking), 274G (switched 2740 Model 1 with Checking), 274H (switched 2740 Model 1 with Transmit Control), 274I (2740 Model 2 with Checking), 274J (2740 Model 2 without Checking), 2741, 2760, 277A (polled 2770), 277B (non-polled 2770), 278A (polled 2780), 278B (non-polled 2780), 7770, 113A (polled 1130), 113B (non-polled 1130), 202A (polled Model 20), 202B (non-polled Model 20), 83B3, 115A (Western Union Plan 115A outstations on a nonswitched network), 3335 (AT&T 33/35 Dial), WTTY (World Trade telegraph terminals), S36B (non-polled System/360).

QUEUES=form

Function: Specifies where the message queues are to be maintained.
Default: None. This operand must be specified.
Format: DR, DN, MO, MN or MR.
Notes: For a discussion of this topic, see *Message Queues Data Sets.*

If queuing is by terminal, this operand must be specified for all TERMINAL macros for a station on the line. If queuing is by line, this operand must be specified for the first

TERMINAL macro coded for a station on the line, but may be omitted for subsequent TERMINAL macros for stations on the line.

DR specifies reusable disk queues.
DN specifies nonreusable disk queues.
MO specifies main-storage-only queues.
MR specifies main-storage queues with backup on reusable disk.
MN specifies main-storage queues with backup on nonreusable disk.

If MO is specified, the distribution list, multiple routing and REDIRECT facilities should be used with care, since one extra buffer is required to accommodate every destination other than the original destination.

If the form of data set specified by this operand does not correspond to a related message queues data set specified in the DCB, the TCAM system terminates abnormally.

If MO, MR or MN is specified, the MSUNITS= operand of the INTRO macro must specify a non-zero integer; otherwise, the TERMINAL macro does not assemble properly and an MNOTE is generated.

DIALNO={chars / NONE}

Function: Specifies the telephone number of the station.
Default: None. Specification optional.
Format: chars or NONE. *chars* is a decimal field with no framing characters.
Notes: This operand tells TCAM whether a station is on a switched or a nonswitched line, and it must be specified for switched stations. *chars* is the telephone number of the station.

DIALNO=chars must be specified if the CINTVL= operand of this macro is specified. DIALNO=NONE specifies that this station is on a switched line, but the computer may not initiate calls to it. DIALNO=NONE must be specified if the Transmission Control Unit for the line over which contact is to be established with the station does not have the Auto Call feature and should be specified if Inward WATS lines are to be used to best advantage.

If this operand is omitted, the station for which this TERMINAL macro is coded is assumed to be on a nonswitched line.

ADDR=chars

Function: Specifies the addressing characters for the station, or specifies the end-to-end control sequence for switched or nonswitched point-to-point 2770 or 2780 stations.
Default: None. Specification optional.
Format: Unframed hexadecimal equivalent of the appropriate transmission code representation.
Notes: Addressing characters are used by the central computer to inform a station that the computer wishes to send it a message. For information on the addressing characters for a specific station, see the hardware manual for that station.

If a station is assigned an ID sequence rather than addressing characters, this operand is not coded; the ID sequence is entered in the invitation list (see the discussion of the INVLIST macro).

This operand may also specify the end-to-end control sequence for a point-to-point 2770 or 2780 station. For information on the end-to-end control sequence, see the appropriate hardware manual. The end-to-end control sequence is specified by writing the equivalent of the appropriate transmission code representation, and must be immediately preceded by the line-control character STX and immediately followed by the line-control character ETB.

LEVEL=(integer,...)

Function: Specifies the permissible priority levels that may be used in the header of a message destined for this station.
Default: None. Specification optional.
Format: Unframed decimal integer.
Maximum: 255.
Notes: The levels must be specified in increasing order. For instance, if the messages being sent to a certain station can have priorities of 1, 9 or 11, the LEVEL= operand for this station would be coded LEVEL= (1,9,11). If queuing is by line rather than by terminal, the priority levels specified in the first TERMINAL macro coded for a station

on the line will apply to all stations on that line; in this case, the LEVEL= operand of subsequent TERMINAL macros for the same line is ignored.

For more information on message priority, see the discussion of the PRIORITY macro and *Message Priority and Queuing* in this chapter.

CLOCK=time

Function: Specifies the time of day that the computer should initiate contact with a switched station.
Default: None. Specification optional.
Format: Two decimal integers for the hours, immediately followed by two decimal integers for the minute. Framing characters may not be specified.
Maximum: 2359 (i.e., 23 for the first field, 59 for the second).
Notes: If this operand is specified, CINTVL= must be omitted and DIALNO= must specify the dial digits to be used. If CLOCK= and CINTVL= are both omitted, the computer does not periodically initiate contact with this switched station. When CLOCK= is specified, the only time that the switched station will be sent messages is when the computer initiates contact with it (at the time of day specified by this operand). If the station calls in at any time other than that specified by this operand, it may enter messages but will *not* be sent any messages by the computer (except that a station locked to an application program will get its lock responses). This operand is used to take advantage of low toll times.

CINTVL=integer

Function: Specifies the period of time following which the computer should initiate contact with a switched station, if neither the station nor the computer called the other during this period.
Default: None. Specification optional.
Format: Unframed decimal integer.
Maximum: 65535
Notes: The interval is restarted at the termination of each call from the station, or when the computer calls the station to send its messages. If CINTVL= is specified, DIALNO= must also be specified, and CLOCK= must be omitted. The first interval starts when the line group data set for this line is opened. This operand can be used to take advantage of Outward WATS.

BUFSIZE=integer

Function: Specifies the buffer size for outgoing messages destined for this station, and overrides the BUFSIZE= operand of the line group DCB macro.
Default: None. Specification optional.
Format: Unframed decimal integer greater than 32.
Maximum: 65535
Notes: If this operand is omitted, the buffer size specified in the line group DCB macro is used.

ALTDEST=entry

Function: Specifies the alternate destination to which a message on a reusable disk queue is sent at the time the zone containing the message is serviced for reuse.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols, and specify the name of any single, group, or process entry in the terminal table capable of accepting messages. Framing characters must not be specified.
Notes: See *Reusable Disk Queues* for a description of servicing for reuse. The name of the TERMINAL macro for which this ALTDEST= operand is being coded may be specified to prevent the message being discarded from a reusable queue. If the ALTDEST= operand is omitted, messages in a reusable disk queue may be written over and lost to the system with no error indication being made. This operand is ignored unless either QUEUES=DR or QUEUES=MR is specified in this TERMINAL macro.

BFDELAY=integer

Function: Specifies the number of seconds of delay to be used before another message block is sent to a buffered station (to avoid sending another message block while the hardware buffer is still emptying the previous block of data).
Default: None. Specification optional.
Format: Unframed decimal integer.
Maximum: 65535
Notes: *integer* should specify the average time needed to empty the hardware buffer. This may be computed from the number of characters in the hardware buffer and the rate at which characters are transferred from the buffer to the terminal component.

This operand must be coded only for IBM 2740 Model 2 and multipoint IBM 2770 stations and should not be coded for any other station. For information on TCAM's buffering feature, see *Transmission Priority for Nonswitched Polled Stations Using TCAM's Buffering Feature* in this chapter.

The BFDELAY= operand must either be included in all TERMINAL macros for stations on the same line, or omitted from all TERMINAL macros for stations on the line. When this operand is coded, queuing by station and send priority should also be specified. If stations using a buffer delay are intermixed with nonbuffered stations on the same line (this can be done because BSC stations are compatible), BFDELAY=0 should be specified in the TERMINAL macros for the nonbuffered stations. The BFDELAY= operand should not be specified for start-stop or BSC stations on switched or point-to-point lines.

NTBLKSZ=(blocksize,subblocksize)

Function: Specifies blocking factors for outgoing messages in nontransparent mode directed to this station.
Default: None. Specification optional.
Format: Unframed decimal integer.
Maximum: For *blocksize,* 65535. For *subblocksize,* 255.
Notes: *blocksize* is the number of bytes in each block of data in nontransparent mode for messages directed to this station, when the MSGFORM macro is coded in the outheader subgroup handling these messages.

blocksize is used when LC=OUT is specified in the STARTMH macro to indicate where EOB or ETB line-control characters are to be inserted in outgoing messages. If a blocksize of 100 were specified, an EOB or ETB would be inserted after every 100 characters in the message, provided that the message were handled by an outheader subgroup that contains a MSGFORM macro. The value specified here may be overridden by coding the BLOCK= operand of the MSGFORM macro; if the *blocksize* suboperand is omitted from the TERMINAL macro, MSGFORM may still be used to specify the blocking factor. The character inserted is not considered part of the block.

subblocksize is the number of bytes in each subblock of data in nontransparent mode for messages directed to this BSC station. It is used when LC=OUT is specified in the STARTMH macro to indicate where ITB line control characters are to be inserted in outgoing messages. If a subblocksize of 100 were coded, an ITB would be inserted after every 100 characters in the message, provided that the message were handled by an outheader subgroup that includes a MSGFORM macro. The value specified here may be overriden by coding the SUBBLOCK= operand of the MSGFORM macro; if the *subblocksize* suboperand is omitted from the TERMINAL macro, MSGFORM may still be used to specify the number of bytes per subblock. The ITB inserted is not considered part of the block.

TBLKSZ=integer

Function: Specifies the number of bytes in each block of data for outgoing messages in transparent mode.
Default: None. Specification optional.
Format: Unframed decimal integer.
Maximum: 65535
Notes: The appropriate line control sequence is transmitted after each number of bytes of data specified by *integer,* provided that the MSGFORM macro is coded in the outheader subgroup handling this message, and provided that SENDTRP=YES is coded in MSGFORM. The value specified here may be overridden by coding the BLOCK= operand of the MSGFORM macro; if the TBLKSZ= operand is omitted from the TERMINAL macro, MSGFORM may still be used to specify the blocking factor for outgoing messages in transparent mode.

OPDATA=(data,...)

Function: Specifies the actual data to be inserted in the set of option fields assigned to this station (see the discussion of the OPTION macro), and also specifies which option fields are not to be created for this station.
Default: None. Specification optional.
Format: The maximum length and type of data specified for each option field must correspond to the length and type specified by the OPTION macro that defines the field, and the order in which the data for each field is specified must correspond to the order in which the OPTION macros are specified. Framing characters are not used.

Notes: When specifying option fields for a particular station, the user may omit the last several option fields defined by OPTION macros by merely closing the parentheses after the data for the last field he wishes to define. A comma is used to:

1. delimit the data for each field;
2. indicate that no data is specified for the first or an intermediate field defined by an OPTION macro;
3. indicate that the OPDATA operand is to be continued (if specified immediately preceding the right parenthesis—see note below).

The user must specify either data and a comma, or a comma alone for the first and each intermediate field (except the last) that is specified by an OPTION macro (with one exception—see the note below). A comma alone is coded if a field other than the last is not to be defined for this station. If the last field is not to be defined, no data is coded for the field and the comma is also omitted. Framing characters (X or C and quotes) are not coded.

Example:
Assume that four OPTION macros have been coded. If the user wants to specify all four fields for a particular station, line, or application program, he would code the OPDATA= operand of the TERMINAL or TPROCESS macro as follows:

```
,OPDATA=(field1,field2,field3,field4)
```

where *field1, field2, field3,* and *field4* represent the actual initial data to be inserted into each of the four option fields. If only *field1* and *field4* are to be implemented for this station, line, or application program, the user would code

```
,OPDATA=(field1,,,field4)
```

If only *field1, field2,* and *field3* are to be implemented, the user would code

```
,OPDATA=(field1,field2,field3)
```

If only *field1* is to be implemented, the user would code

```
,OPDATA=(field1)
```

NOTE: Because the operand field of a macro is limited to 255 characters, TCAM provides a facility to specify additional OPDATA= parameters if necessary. A comma placed as the last character of the OPDATA= operand—i.e.,

```
OPDATA=(data,data,...data,)
```

indicates a continuation of the OPDATA= operand. The next source statement would then be coded

```
symbol TERMINAL OPDATA=(data,...)
```

where

symbol is the name specified on the TERMINAL macro that specified the continuation.

There is no limit (other than the number of OPTION fields defined) on the number of continuation statements that may be used.

SECTERM= {**YES**} / {**NO**}

Function: Specifies whether this station may be considered a secondary operator control station.
Default: SECTERM=NO
Format: YES or NO.
Notes: If YES is specified, operator commands will be recognized, acted upon and the appropriate response returned to the station. The station for which SECTERM=YES is specified must be on a nonswitched line and must be able to both enter and accept messages. If a station other than the system console is to be the primary operator control station, SECTERM=YES must be specified for that station's TERMINAL macro.

COMP={YES / NO}

Function: Specifies whether or not this TERMINAL macro is being used to define a component of a station defined by another TERMINAL macro.
Default: COMP=NO
Format: YES or NO.
Notes: If this operand is coded COMP=YES, then the TERMINAL macro is for a component. If the operand is omitted or COMP=NO is coded, then the macro is not for a component.

For guidelines on coding this operand, see *Coding the Terminal Macro for a Component* in this chapter.

UTERM={YES / NO}

Function: Specifies whether or not this TERMINAL macro is being used to define a line entry in the terminal table.
Default: UTERM=NO
Format: YES or NO.
Notes: If this operand is coded UTERM=YES, then the TERMINAL macro is for a line. If the operand is omitted, or if UTERM=NO is coded, then the macro is not for a line.

For information on coding this operand, see *Coding the TERMINAL Macro for a Line* in this chapter.

Coding the TERMINAL Macro for a Component

If the COMP= operand of a terminal macro is coded COMP=YES, then the TERMINAL macro is one defining a component of a station defined by another TERMINAL macro. A TERMINAL macro need be issued for a component only if messages may be directed to more than one component of a station by means of appropriate addressing characters. If addressing characters are not used, a TERMINAL macro for a component is unnecessary. If a message can be sent to only one component of a terminal assigned addressing characters, that component may be specified by coding the appropriate addressing characters in the ADDR= operand of the TERMINAL macro for the terminal. For an IBM 1050 terminal assigned addressing characters, for example, the second addressing character identifies the component that is to receive the message. If only one component is to receive messages, that component's selection character may be entered as the second addressing character in the ADDR= operand of the TERMINAL macro for the terminal, and no TERMINAL macro need be issued for the component. If more than one component of a station is to be specifically addressed by means of addressing characters, then one or more component TERMINAL macros must be issued; these should immediately follow the TERMINAL macro for the station to which the components belong.

The following operands of the TERMINAL macro are meaningful if the macro is issued for a component:

ADDR=chars

Specifies the addressing characters for this component.

ALTDEST=entry

Specifies the alternate destination to which a message on a reusable disk queue is sent at the time the zone containing the message is serviced for reuse (see *Reusable Disk Queues* for a description of this servicing). Any terminal, component, or process entry for a device capable of accepting messages may be specified. If the operand is omitted, messages in a reusable disk queue may be written over and lost to the system with no error indication being made.

SECTERM={YES / NO}

Specifies whether replies to operator commands entered at this station are to be sent to this component. If so, this component must be represented in the invitation list for this line. If the station is polled, the operator command must have been entered in response to polling characters associated in the invitation list with an entry having the same name as the name of this terminal entry. (However, the two entries having the same name need not refer to the same device—the polling characters could poll a card reader, for example, while the addressing characters might address a printer).

NTBLKSZ=(blocksize, subblocksize)

Specifies blocking factors for outgoing messages in nontransparent mode directed to this station. *blocksize* and *subblocksize* have the same meanings as those described above in the discussion of the TERMINAL macro for a station.

TBLKSZ=integer

Specifies the number of bytes in each block of data for outgoing messages in transparent mode directed to this component. This operand is similar to the TBLKSZ= operand for the TERMINAL macro for a station, described above, and may be overridden by coding the BLOCK= operand of the MSGFORM macro, and specifying SENDTRP=YES in MSGFORM.

BUFSIZE=integer

Overrides the buffer size specified by the BUFSIZE= operand of the line group DCB macro, but only for buffers containing outgoing messages destined for this component. If this operand is omitted, the buffer size specified in the line group DCB macro is used.

OPDATA=(data,...)

Specifies the actual data to be inserted in the set of option fields assigned to this component (see the discussion of the OPTION macro), and also specifies which option fields are not to be created for this component. The description of the OPDATA= operand of the TERMINAL macro for a station also applies to the OPDATA= operand of the TERMINAL macro for a component.

COMP={YES}
     {NO }

Specifies whether this TERMINAL macro is for a component. COMP=YES indicates that this TERMINAL macro is for a component.

Coding the TERMINAL Macro for a Line

A TERMINAL macro whose UTERM= operand is coded UTERM=YES causes information to be included in the terminal table for a line to switched stations that do not uniquely identify themselves when calling the computer.

As a general rule, a switched line requires its own TERMINAL macro if any stations that do not always uniquely identify themselves call the computer on that line. If all stations calling in on a switched line always uniquely identify themselves, no TERMINAL macro is required for that line. The following considerations apply when deciding whether a particular switched line requires its own TERMINAL macro (see also Figure 1, which summarizes these considerations):

1. A TCAM audio line (i.e., a line connected to an IBM 7770 Audio Response Unit, Model 3) requires its own TERMINAL macro.
2. A switched line to BSC stations that are all assigned unique ID sequences does not require its own TERMINAL macro. For such a line, the user should enter each station's name and ID sequence, and the CPU ID sequence, in the appropriate operands of the INVLIST macro for the line (see the discussion of the INVLIST macro).
3. If none of the stations on a line ever dial the computer, the line needs no TERMINAL macro. Terminal names and invitation characters are coded in the INVLIST macro (see the discussion of the INVLIST macro).
4. For a switched line to stations other than those described in (2) and (3) above, a TERMINAL macro specifying UTERM=YES must be coded unless all messages entered by stations on the line have origin fields in their message header and are processed by a Message Handler subgroup containing an ORIGIN macro (see the discussion of the ORIGIN macro). For lines to stations that enter only messages having origin fields, see (5). When a TERMINAL macro is coded for a line, the name of the macro is entered together with the invitation characters for stations on the line in the appropriate operand of the INVLIST macro for the line (see the discussion of the INVLIST macro).

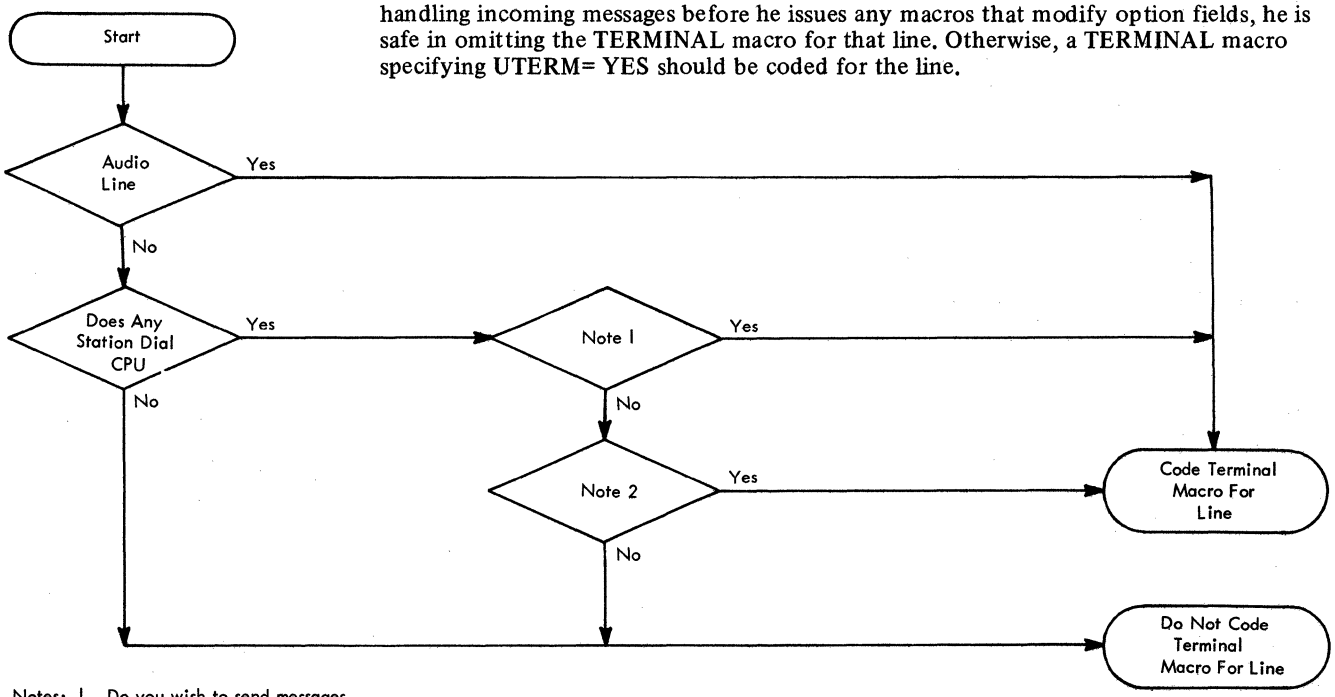
5. For a switched line to stations other than those described in (2) and (3), if all messages entered on the line have valid origin fields in their message headers and are processed by a Message Handler subgroup containing an ORIGIN macro, then a TERMINAL macro may be specified for that line at the option of the user. If the TERMINAL macro is specified, the user must enter its name as part of the *entry* operand of the INVLIST macro for the line; otherwise the name of a TERMINAL macro for a station on the line is entered as part of the INVLIST entry. In either case, one INVLIST entry is coded for each series of invitation characters used by a station on the line (see the discussion of the INVLIST macro).

In order to decide whether to code a TERMINAL macro for a line under Case 5, the user must first understand the function of the ORIGIN macro and the origin field of the message header (discussed in the chapter *Designing the Message Handler*) and must also understand the function of the OPTION macro (discussed in the present chapter). Then he should consider the following paragraphs.

The TERMINAL macro has an optional OPDATA= operand. If a TERMINAL macro is coded for a switched line, when a station on that line dials the computer the option fields associated with the line entry are accessed and possibly modified by Message Handler macros until an ORIGIN macro is encountered in the Message Handler.

The ORIGIN macro establishes the identity of the calling station; once identity has been established, the option fields associated with the terminal entry for the station calling in are accessed and possibly modified by Message Handler macros. If no line entry is created for a switched line, then the option fields assigned to the representative station named in the INVLIST macro operand are updated by the Message Handler macros until the ORIGIN macro is encountered; once ORIGIN has established the identity of the calling station, that station's option fields are updated by macros following ORIGIN in the Message Handler.

Thus, if a user assigns no option fields to the stations on a switched line, or if he does assign option fields but issues his ORIGIN macro in the Message Handler subsections handling incoming messages before he issues any macros that modify option fields, he is safe in omitting the TERMINAL macro for that line. Otherwise, a TERMINAL macro specifying UTERM= YES should be coded for the line.



Notes: 1. Do you wish to send messages to unknown stations on the line?
2. Do you wish to update any option fields on a line basis?

Figure 1. Chart for Deciding Whether a TERMINAL Macro Should be Coded for a Switched Line.

All TERMINAL macros for lines in a line group must be arranged according to ascending relative line number. The TERMINAL macro for a particular line must immediately precede all TERMINAL macros for stations on that line.

Example:
The TERMINAL macros for three switched lines in a line group, where each line has three terminals associated with it, would be arranged in the following order:

- TERMINAL macro for relative line 1 (UTERM=YES)
- TERMINAL macro for a terminal on line 1
- TERMINAL macro for another terminal on line 1
- TERMINAL macro for a third terminal on line 1
- TERMINAL macro for relative line 2 (UTERM=YES)
- TERMINAL macro for a terminal on line 2
- TERMINAL macro for another terminal on line 2
- TERMINAL macro for a third terminal on line 2
- TERMINAL macro for relative line 3 (UTERM=YES)
- TERMINAL macro for a terminal on line 3
- TERMINAL macro for another terminal on line 3
- TERMINAL macro for a third terminal on line 3

It may be that some lines in a line group have TERMINAL macros coded for them and others do not. In this case, arrange the TERMINAL macros for the stations on the lines in groups according to ascending relative line number, and place each TERMINAL macro for a line immediately in front of the group of TERMINAL macros for stations on that line.

Example:
The TERMINAL macros for three switched lines in a line group, where each line has two terminals associated with it, and line 2 has no TERMINAL macro coded for it, would be arranged in the following order:

- TERMINAL macro for relative line 1 (UTERM=YES)
- TERMINAL macro for a terminal on line 1
- TERMINAL macro for another terminal on line 1
- TERMINAL macro for a terminal on line 2
- TERMINAL macro for another terminal on line 2
- TERMINAL macro for relative line 3 (UTERM=YES)
- TERMINAL macro for a terminal on line 3
- TERMINAL macro for another terminal on line 3

The following operands of the TERMINAL macro are relevant when the macro is specified for a line:

DCB=dcbname
RLN=integer
TERM=type
ADDR=chars
OPDATA=data
UTERM={YES / NO}

The DCB=, RLN=, and TERM= operands are the same as those given above for a TERMINAL macro for a station.

The ADDR= operand need be coded in the TERMINAL macro for a line only when a station that does not identify itself (by means of an origin field in the message header, as checked by an ORIGIN macro in the Message Handler) when calling the computer may call in on this line and either:

(a) cause a response message to be sent back to the originating station by means of a MSGGEN macro in the Message Handler, or
(b) place itself in lock mode (see the description of the LOCK macro) in order to await a response message from an application program.

The ADDR= operand of the TERMINAL macro for a station must be coded unless that station fails to identify itself after calling the computer and is restricted in usage to (a) or (b) above. If the ADDR= operand of the TERMINAL macro for a line is coded, all stations on the line must have identical addressing characters.

The OPDATA= operand specifies the data to be inserted in the set of option fields assigned to this line. The operand is coded in the same way as the OPDATA= operand of a TERMINAL macro for a station.

When a station on this line dials the computer, the option fields assigned to the line are accessed and modified by Message Handler macros until an ORIGIN macro in the Message Handler establishes the identity of the calling station; once identity is established, the option fields assigned to the station calling in are updated by macros following the ORIGIN macro in the Message Handler. When the computer calls a station, only the option fields assigned to the station may be updated.

The UTERM= operand specifies whether this TERMINAL macro is for a line. If UTERM=YES is coded, this TERMINAL macro is a line. If the operand is omitted or coded UTERM=NO, this TERMINAL macro is not for a line.

TLIST Macro Instruction

The TLIST macro:

- Defines a cascade list entry or distribution list entry in the terminal table,
- Is optional among macros defining the terminal table.

The TLIST macro causes the name of a list of single, group, or process entries in the terminal table, together with information about the entries in the list, to be included as an entry in the terminal table.

A distribution or cascade list consists of the names of single, group, or process entries in the terminal table. One TLIST macro must be specified for each list to be created. Stations cannot enter messages using a distribution or a cascade list.

When a message contains the name of a distribution list as a destination code, TCAM sends the message via separate transmissions to each station or application program indicated by an entry in the list. Each entry in the list must have a corresponding single, group, or process entry in the terminal table. When a message contains the name of a cascade list as a destination code, TCAM places the message on the destination queue for that valid destination in the list that has the fewest messages waiting to be sent to it. If several destinations have the same number of messages, the message is queued for the first such destination.

The TLIST macro provides the initial contents for all fields in the list entry.

Name	*Operation*	*Operand*
symbol	TLIST	TYPE={D|C}, LIST=(entry,entry,...)

symbol

Function: Specifies the name of the list.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

TYPE={D|C}

Function: Specifies whether the list is a distribution or a cascade list.
Default: None. This operand is required.
Format: D or C.
Notes: C specifies a cascade list. D specifies a distribution list.

LIST=(entry,entry,...)

Function: Specifies the actual entries in the distribution list or cascade list being created.
Default: None. This operand is required.
Format: Each entry is the name of a single, group, process, or cascade list entry in the terminal table. If TYPE=D, at least two entries must be specified. If TYPE=C, only one entry is required.
Notes: The name of a distribution list entry in the terminal table may not be specified as an entry in a distribution list. If the list being created is a distribution list, it may contain the name of one or more cascade list entries. If it is a cascade list, it may not contain the name of a cascade list entry.

Because of the limitation of 255 characters in a macro operand, a facility is provided to specify additional TLIST entries if necessary. A comma placed as the last character of the

entries operand indicates a continuation of the list. The next source statement would then be coded:

```
symbol TLIST LIST=(entry,...)
```

where

symbol is the TLIST name as specified on the previous TLIST macro that specified the continuation. There is a limit of 32767 entries in a distribution or cascade list.

TPROCESS Macro Instruction

The TPROCESS macro:

- Serves as part of the interface between the MCP and an application program.
- Creates a terminal table entry for a queue associated with an application program.
- Is optional among macros defining the terminal table.

The TPROCESS macro causes the name of a queue for an application program and associated information to be included as an entry in the terminal table. The entry produced is a process entry.

One TPROCESS macro must be included for each destination queue to which an application program can direct a GET or READ macro, and at least one must be included for each process entry to which a PUT or WRITE macro may be directed.

An operand of the TPROCESS macro specifies the name of a process control block (PCB), which is used to establish communication between a Message Handler and application programs. (The PCB is created by means of a PCB macro.)

An operand of TPROCESS enables the user to specify one alternate destination to which the message may be sent in certain circumstances.

The user may specify that checkpointing of the application program is to be synchronized with that of the Message Control Program. Synchronization of OS with TCAM checkpoints is discussed in the chapter *Writing TCAM-Compatible Application Programs.*

The user also specifies the initial contents of the option fields for the process entry in the terminal table.

The TPROCESS macro helps connect an application program to the Message Control Program. The GET and PUT or READ and WRITE macros issued in an application program each specify the name of a data control block created by a DCB macro issued in the application program. The DCB macro specifies (by means of its DDNAME= operand) a DD card. The QNAME= parameter of the DD card names a process entry. The *pcbname* operand of the TPROCESS macro creating this entry specifies a process control block. The MH= operand of the PCB macro creating the process control block specifies the Message Handler that handles messages directed to and received from the application program.

The TPROCESS macro has the following format:

Name	*Operation*	*Operand*
procname	TPROCESS	PCB=pcbname[,QUEUES=form] [,ALTDEST=entry] [,CKPTSYN={YES / NO}] [,SECTERM={YES / NO}] [,RECDEL=delimiter] [,LEVEL=(integer,...)] [,OPDATA=(data,....)]

procname

Function: Specifies the name of the process entry in the terminal table.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).
Notes: The name must be specified and must be the same as that entered in the QNAME= parameter of the DD statement associated with the DCB macro for an application program.

PCB=pcbname

Function: Specifies the name of the process control block that defines buffers, etc., to handle messages queued to this process entry.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: The process control block is created by a PCB macro. All TPROCESS macros issued for the same application program must have the same PCB.

QUEUES=form

Function: Specifies where the message queues containing messages for the application program are to be maintained.
Default: None. Specification optional.
Format: DR, DN, MO, MR or MN.
Notes: DR specifies reusable disk queues. DN specifies nonreusable disk queues. MO specifies main storage only queues. MR specifies main storage queues with reusable disk backup. MN specifies main storage queues with backup on nonreusable disk. If the form of data set specified by this operand does not correspond to a related message queues data set defined by a DCB macro, the TCAM system terminates abnormally. By omitting the QUEUES= operand, the user specifies that this process entry is for PUTs or WRITEs from an application program.

If MO, MR or MN is specified, the MSUNITS= operand of the INTRO macro must specify a non-zero integer; otherwise, the TPROCESS macro does not assemble properly and an MNOTE is generated.

ALTDEST=entry

Function: If this process entry is for GETs or READs issued by an application program, this operand specifies the alternate destination to be sent at the time the zone containing the message is being serviced for reuse. If this process entry is for PUTs or WRITEs from an application program, this operand specifies the destination to which replies to operator commands issued by the application program are sent.
Default: None. Specification optional.
Format: The name of any single, group, or process entry in the terminal table.
Notes: The entry specified may be the one created by the TPROCESS macro, preventing the message from being discarded from a reusable queue. If this operand is omitted for a GET or READ process entry, the message may be overlaid in a reusable queue and lost to the system. The operand is ignored unless QUEUES=DR or QUEUES=MR is specified for the TPROCESS macro.

For a PUT or WRITE entry, the destination may be a station named by a TERMINAL macro, or it may be an application program represented by a TPROCESS macro.

CKPTSYN={YES / NO}

Function: Specifies whether the destination queue to which the application program directs its GETs or READs is to be purged of serviced messages at restart.
Default: CKPTSYN=NO
Format: YES or NO.
Notes: CKPTSYN=YES specifies that no purging of the queue is to be performed. If an OS checkpoint of the application program is used in synchronization with the TCAM checkpoint, CKPTSYN=YES should be specified. If this operand is omitted, the queue is scanned and updated at restart. When synchronization is not specified, operation following restart resumes with the first unserviced message for the queue (a message is considered serviced when a GET or READ is issued for the next message from the queue and that next message is placed on the queue). The first unserviced message is determined in the scan of the message queue done at restart time. When not using synchronization with an OS checkpoint, it is necessary to check for one duplicate message upon restart (i.e., the message being processed when failure occurred).

For more information on TCAM's checkpoint facility, see the chapter *Using TCAM Service Facilities.* Coordination of OS and TCAM checkpoints is discussed in the chapter *Writing TCAM-Compatible Application Programs.*

SECTERM={YES / NO}

Function: Specifies whether the application program may be considered a secondary operator control station (so that operator commands may be sent to TCAM from the application program by means of a PUT or WRITE macro).
Default: SECTERM=NO
Format: YES or NO.

Notes: This operand is meaningful only if this process entry is associated with a PUT or WRITE macro, and is ignored if coded for a process entry associated with a GET or READ macro. If this process entry is to be the primary operator control station, SECTERM=YES must be specified for the entry.

RECDEL=delimiter

Function: For a process entry associated with a GET or READ macro this operand specifies a one-byte non-zero hexadecimal value used to delimit a record for the application program. For a process entry associated with a PUT or WRITE macro, this operand specifies a value to be inserted at the end of each variable-length record returned from an application program by means of a PUT or WRITE macro specifying the DCB associated (by means of the QNAME= operand of its DD card) with the process entry.
Default: None. Specification optional.
Format: A single unframed hexadecimal character.
Notes: This character may be inserted periodically into a TCAM buffer by means of a MSGEDIT macro whose DATA operand is coded DELIMIT, for a process entry associated with a GET or READ. If the RECFM= operand of the input DCB macro specified by a GET or READ macro in the application program is coded RECFM=V, and if the OPTCD= operand does not have the U suboperand coded in it, the application program GET or READ considers this character to be a record delimiter. The delimiter specified by RECDEL= may be included by the user in the incoming message, or may be inserted by means of a MSGEDIT macro.

For a process entry associated with a PUT or WRITE macro, TCAM automatically inserts the value at the end of each variable-length record. For other than variable length records, this operand is meaningless.

LEVEL=(integer,...)

Function: Specifies the permissible priority levels that may be used in the header of a message enqueued on this process queue.
Default: None. Specification omitted.
Format: Each integer is a decimal integer. The *integer,...* values must be specified in ascending order.
Maximum: 255
Notes: If this operand is omitted, all messages sent to the application program by this process entry are assumed to have zero priority. If the messages being sent to the application program via this process entry can have, for example, priorities of 1, 9 or 11, the LEVEL= operand would be coded LEVEL=(1,9,11).

For more information on message priority, see the discussion of the PRIORITY macro and *Message Priority* in this chapter.

OPDATA=(data,...)

Function: Specifies the actual data to be inserted in the set of option fields assigned to this process entry (see the discussion of the OPTION macro), and also specifies which option fields are not to be created for this process entry.
Default: None. Specification optional.
Format: The maximum length and type of data specified for each option field must correspond to the length and type specified by the OPTION macro that defines the field. The order in which the OPTION macros are specified must correspond to the values of data specified in this operand.
Notes: A comma is used to:
1. delimit the data for each field;
2. indicate that no data is specified for the first or an intermediate field defined by an OPTION macro;
3. indicate that the OPDATA= operand is to be continued (if included immediately preceding the right parenthesis–see note below).

The user must specify either data and a comma, or a comma alone for the first and each intermediate field (except the last) that is specified by an OPTION macro (with one exception–see the note below). A comma alone is coded if a field other than the last is not to be defined for this line. If the last field is not to be defined, no data is coded for the field and the comma is also omitted. Framing characters (X or C and quotes) are not coded.

NOTE: When specifying option fields for a particular process entry, the user may omit the last several option fields defined by OPTION macros by merely closing his parentheses after the data for the last field he wishes to define.

Example:
Assume that four OPTION macros have been coded. If the user wants to specify all four fields for a particular station, line, or application program, he would code the OPDATA= operand of the TERMINAL or TPROCESS macro as follows:

```
,OPDATA=(field1, field2, field3, field4)
```

where *field1, field2, field3,* and *field4* represent the actual initial data to be inserted into each of the four option fields. If only *field1* and *field4* are to be implemented for this station, line or application program, the user would code

```
OPDATA=(field 1,,,field 4)
```

If only *field1, field2,* and *field3* are to be implemented, the user would code

```
,OPDATA=(field1,field2,field3)
```

If only *field1* is to be implemented, the user would code

```
,OPDATA=(field1)
```

A message processed by an application program and then sent to a destination station must be handled by two sets of incoming and two sets of outgoing MH subgroups. Macros issued in the incoming subgroups handling messages coming in from a station update the option fields assigned to that station. Macros issued in the outgoing subgroups handling messages for the application program update the option fields assigned to the process entry associated with the GET or READ macro that obtains the messages for processing. Macros issued in the incoming subgroups handling messages from an application program update the option fields assigned to the process entry associated with the PUT or WRITE macro that returns messages from the application program to the MCP. Macros issued in outgoing subgroups handling messages being sent to a destination station update the option fields assigned to that station. (For a description of which Message Handler subgroups are required when there is an application program, see *Message Flow through a Message Handler* in the chapter *Designing a Message Handler.* For a discussion of the interface between the MCP and the application program see the introduction to *Writing TCAM-Compatible Application Programs.*)

NOTE: Because the operand field of a macro is limited to 255 characters, TCAM provides a facility to specify additional OPDATA= parameters if necessary. A comma placed as the last character of the OPDATA= operand—i.e., OPDATA=(data,data,...data,) indicates a continuation of the OPDATA= operand. The next source statement would then be coded

```
symbol TPROCESS OPDATA=(data,...)
```

where

symbol is the process-entry name as specified on the TPROCESS macro that specified the continuation. There is no limit (other than the number of option fields defined) on the number of continuation statements used.

LOGTYPE Macro Instruction

The LOGTYPE macro:

- Initializes for using TCAM's logging facility,
- May not be omitted if TCAM's logging facility is to be used for logging complete messages, and is unnecessary if segments are logged,
- If coded, must be specified among the macros defining the terminal table and must not be the last such macro.

The LOGTYPE macro initializes TCAM's logging facility by specifying:

1. the name of the data control block for the log data set,
2. the buffer size used to handle messages to be logged,
3. the location of the data set (on disk or in main storage);

TCAM's logging facility is discussed in *Using TCAM Service Facilities*. The description of the LOG macro contains information on when LOGTYPE should be specified.

NOTE: A LOGTYPE macro *must not* be coded as the last macro defining the terminal table. No more than one LOGTYPE macro should be coded for a log data set.

Name	*Operation*	*Operand*
typename	LOGTYPE	dcbname, BUFSIZE=size [,QUEUES=form]

typename

Function: Specifies the name of the LOGTYPE macro and is the same as the *typename* operand of a LOG macro.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

dcbname

Function: Specifies the name of the data control block for the log data set.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: This name must be the same as the name of the DCB macro specifying the log data set.

BUFSIZE=size

Function: Specifies the size of the buffers to be used to handle messages destined for the logging medium.
Default: None. This operand is required.
Format: Unframed decimal integer greater than 32.
Maximum: 65535

QUEUES=form

Function: Specifies where the messages are to be queued while awaiting transfer to the logging medium.
Default: None. This operand is required.
Format: DR, DN, MO, MR or MN.
Notes: DR specifies reusable disk queues. DN specifies nonreusable disk queues. MO specifies main storage only queues. MR specifies main storage queues with reusable disk backup. MN specifies main storage queues with backup on nonreusable disk.

Maintaining Orderly Message Flow

Thus far, this chapter has described how to define control areas needed by TCAM for line control, and how contact is established for the purposes of invitation and selection. This section describes how TCAM maintains an orderly message flow between the central computer and remote stations.

Among the factors influencing the flow of messages within a TCAM system are message priority and queuing, and transmission priority. *Message priority* refers to the order in which messages are sent over a line or to an application program, relative to each other. This order depends upon the priorities assigned to individual messages by the user as specified by a priority field in the message header and a PRIORITY macro, and upon the type of queuing (whether by line or by terminal) specified by the QBY= operand of the TERMINAL macro. *Transmission priority* refers to the relative order in which messages are sent to and received from a station or stations on a line. The transmission priority (send, equal, or receive) for a nonswitched station is specified by the CPRI= operand of the line group DCB macro. For switched stations, CPRI=S (indicating send priority) must always be specified.

Message priority and queuing, and transmission priority are not the only factors influencing TCAM message flow; two other factors are the manner in which calls are made between the computer and a switched station, and the system interval.

The remainder of this chapter is devoted to discussions of message priority and queuing, transmission priority, calls between the computer and a switched station, and the system interval.

Message Priority and Queuing

To determine how to assign priorities to messages in a TCAM system, see the descriptions of the PRIORITY macro and of the LEVEL= operand of the TERMINAL and TPROCESS macros. In this section, we shall be concerned with a practical description of what message priority means in a TCAM system; more specifically, given a certain number of messages having different priorities, which are awaiting transmission to destination stations on a certain line, or which are enqueued on a process queue for an application program, we shall describe the order in which these messages are sent, relative to each other. This order depends upon three variables:

- Whether queuing is by line or by terminal,
- The relative order in which the messages are received at the destination queue,
- What priorities the messages are assigned.

Messages whose destinations are stations may be queued by destination terminal or by destination line. The user specifies the type of queuing he wants by the QBY= operand of the TERMINAL macro. When outgoing messages are queued by line, one message queue is created for a line, and messages destined for all stations on the line are placed on this queue. (The incoming group of a Message Handler generally determines the destination of a message by a FORWARD macro.) Messages are taken off the queue and sent to stations on the line on a first-ended first-out (FEFO) basis within priority groups. That is, messages on the queue that have a higher message priority (as specified in the message header or assigned by a PRIORITY macro) are sent before messages having a lower priority; when messages have the same priority, the one whose final segment arrived at the queue first will be sent out first, and the others will be sent out in the order in which their final segments arrived at the queue. (An example of queuing by line is given below.)

Advantages of Queuing by Line

- Queuing by line permits transmission of messages by priority on a line basis to stations on a multipoint nonswitched line; that is, all messages of a given priority on the queue are transmitted before any messages of a lower priority, whether or not the higher-priority messages are destined for two different stations on the line.
- Queuing by line takes less storage space than queuing by terminal. If queuing is by line rather than by terminal, at least 65 bytes are saved for each station after the first on a line, plus about 28 bytes per station after the first for each priority level specified beyond one.

Disadvantages of Queuing by Line

- Queuing by line results in switching between stations on the line rather than maintaining connection with a station.

When outgoing messages are queued by terminal, one message queue is created for each station on a line. All messages queued for a given station are sent before any messages queued for other stations on the line. Messages on a queue are sent to a station on first-ended first-out (FEFO) basis within priority groups. The first message on a queue is the message whose last segment arrived at the queue before the last segment of any other message arrived at the queue. Higher-priority messages are sent before lower-priority messages; when two messages on a queue have equal priority, the one whose final segment arrived at the queue earliest is sent first. For a multipoint line, the relative order in which *queues* of messages are transmitted is also determined on a FEFO basis; the queue containing the message whose incoming transmission over the line was completed first will be sent before any other queue for a station on that line.

Queuing by terminal must be specified for switched stations and for buffered terminals. If switched stations were queued by line, a station that called in would receive not only its messages, but those for all other stations in the line group as well.

Messages whose destination is an application program are placed on a queue for that program, and are removed from it as if they were messages queued by terminal; that is, they are sent to the application program on an FEFO basis within priority groups.

Advantages of Queuing by Terminal

- Queuing by terminal permits transmission of messages by priority on a station-by-station basis; that is, all messages in a given queue for a station on a line are transmitted before any messages in other queues for the remaining stations on the line are

transmitted, whether or not the other queues contain messages having priorities higher than those for the messages being transmitted. Thus, messages for the same station are sent as a group.

Disadvantages of Queuing by Terminal

- Queuing by terminal takes more storage space than does queuing by line.

The orders of sending described above are disrupted when a message segment for which the INITIATE macro has been executed arrives at a destination queue; such a segment is treated as if it were a completed message having the highest priority on the queue, and is sent before any other message on the queue is sent. In addition, no message on the queue may be sent until all segments of the message for which INITIATE was executed have arrived at the queue and been sent to their destination. (See the description of the INITIATE macro.)

Examples:

Assume a multipoint nonswitched line on which are located the following three terminals (each name given corresponds to the *symbol* field of the TERMINAL macro defining that terminal): NYC, BOS, RAL. Nine messages arrive from various remote stations, or perhaps from an application program; these messages are to be routed to the three terminals on this line. Messages 1 through 9 are completely enqueued on a destination queue in the following temporal order:

1 for NYC
2 for NYC
3 for BOS
4 for RAL
5 for RAL
6 for BOS
7 for NYC
8 for RAL
9 for RAL

First, assume that queuing is by line, and that all messages have the same message priority. In this case, the messages are sent out in the same order that they were completely enqueued on the destination queue for the line: 1, 2, 3, 4, 5, 6, 7, 8, 9.

Now, assume that queuing is by terminal and that all nine messages have the same message priority. In this case, the messages are queued

- for NYC in the order 1, 2, 7
- for BOS in the order 3, 6
- for RAL in the order 4, 5, 8, 9

and are sent out in the order 1, 2, 7, 3, 6, 4, 5, 8, 9.

Next, assume that messages 1, 5, and 9 have a message priority of 10, that messages 2, 4, and 7 have a message priority of 30, and that messages 3, 6, and 8 have a message priority of 60.

The messages will be queued by line or by terminal (depending upon which is specified in the TERMINAL macros) in the same order as they would if all messages had the same priority. The order in which they are sent, however, differs from that given above for the case in which all messages have the same priority.

If queuing is by line, the messages are sent in the order 3, 6, 8, 2, 4, 7, 1, 5, 9.

If queuing is by terminal, the messages are sent in the order 2, 7, 1, 3, 6, 8, 4, 5, 9.

Note the following points:

- When messages for stations on a multipoint line are queued by terminal, the order in which the *groups* of messages queued for the individual stations on the line are transmitted with respect to each other depends upon the time that the last segment of the first message on each individual queue arrives at the queue. In the above example the

last segment of the first message on the queue for NYC arrived at its queue before the last segment of the first message on the queue for BOS arrived at its queue, and the last segment of the first message on the queue for BOS arrived at its queue before the last segment of the first message on the queue for RAL. Therefore, all messages queued for NYC are transmitted before any message queued for BOS is transmitted, and all messages queued for BOS are transmitted before any message queued for RAL is transmitted.

- When messages to be sent to stations on a multipoint line are queued by terminal, the order in which the messages queued for an individual station are transmitted is determined by two rules:
  1. All messages having a higher message priority are transmitted before any message having a lower message priority is transmitted;
  2. When messages have equal message priorities, the message whose final segment arrived at the queue earliest is sent first, the message whose final segment arrived at the queue next-earliest is sent second, etc.

When these two rules are in effect, messages are said to be sent out on a first-ended, first-out (FEFO) basis within priority groups.

Messages for stations on point-to-point lines, whether switched or nonswitched, are also transmitted on a FEFO basis within priority groups. (Remember that switched lines are considered to be point-to-point, and that queuing by terminal should always be specified for switched lines.)

Transmission Priority

Transmission priority refers to the relative order in which messages are sent to and received from the stations on a line. Transmission priority is specified on a line-group basis by means of the CPRI= operand of the line group DCB macro.

Transmission priority has a different meaning for each of four configurations of stations supported by TCAM:

1. Polled stations (unbuffered) on a nonswitched point-to-point or multipoint line;
2. Buffered polled stations on a nonswitched multipoint line;
3. Contention stations on a nonswitched point-to-point line;
4. Stations on a switched line.

We shall describe the transmission priority scheme for each of these configurations, in the order given above.

NOTE: TCAM considers a *buffered station* to be one for which the BFDELAY= operand of the TERMINAL macro is coded. A special scheme for transmitting outgoing messages is implemented for such a station (see the description of the BFDELAY= operand of TERMINAL). A station may be defined as *buffered* using the BFDELAY= operand even though no delay is ever taken.

Transmission Priority for Nonswitched Polled Stations

For such stations, the user may specify that sending has priority over receiving (by coding CPRI=S in the line group DCB macro), that receiving has priority over sending (by coding CPRI=R), or that sending and receiving have equal priority (by coding CPRI=E). The meaning of these priorities depends upon whether the line is being polled under the control of the Auto Poll hardware feature, or under the control of the TCAM program polling scheme.

TCAM Program Poll: When this scheme is used, TCAM polls all stations designated as active in the invitation list for an active line. In polling, TCAM begins with the first active station in the list, and invites it to enter a message by sending it polling characters. If the station has a message to enter, it responds by entering the message, following which TCAM polls it again.

If receiving has priority over sending, the cycle of polling and entering is repeated until the first station has no more messages to enter. When TCAM receives a negative response to polling from the first active station in the list, it proceeds to the second active station in the list, and polls it. TCAM continues to poll the second station until the station indicates that it has no more messages to enter, at which time TCAM proceeds to the third station. TCAM proceeds through the list in this fashion until a negative response to polling is received from the last station in the list. At this time, TCAM observes the *invitation delay* specified by the INTVL= operand of the line group DCB macro, or by a

POLLDLAY operator command. During the invitation delay, outgoing messages are sent to stations on the line in the order described in *Message Priority and Queuing.* (If the computer has no messages to send to stations on the line at this time, the invitation delay is observed nevertheless.) Outgoing messages are sent until the delay expires or the destination queues for stations on the line are empty. Upon expiration of the delay, outgoing message transmission ends after the current message is sent, regardless of whether any messages remain queued. As soon as outgoing message transmission ceases, polling and incoming message transmission resume, and the cycle is repeated. It is important to note that if no invitation delay is specified, outgoing message transmission does not occur. If an invitation delay is specified, it must be long enough to accommodate the expected density of outgoing message traffic; too short a delay causes outgoing messages to accumulate on the destination queues for lines or stations in a line group.

If receiving and sending have equal priority, polling and incoming message traffic proceed without interruption until the end of the invitation list is reached. Then outgoing messages (if any are present on the destination queues for the stations on the line) are sent to stations on the line in the order described in the *Message Priority and Queuing* section. Once outgoing transmission begins, it continues until all messages queued for stations on the line have been sent, regardless of whether the user has specified an invitation delay. When all messages for stations on the line have been sent, polling and incoming message traffic resume. Note that, in contrast to the case where receiving has priority over sending, outgoing message transmission occurs whether or not an invitation delay is specified and regardless of the specified length of the delay.

If sending has priority over receiving, outgoing messages (if any are queued for stations on the line) are sent:

1. Each time a negative response to polling is received from a station.
2. Each time an EOT is received from a station, indicating that a complete message has been received.
3. Each time the end of the invitation list is reached.

Outgoing messages are sent in the order described in the *Message Priority and Queuing* section. Once outgoing message transmission begins, it continues until all messages queued for stations on the line have been sent. Note that when sending has priority over receiving, outgoing transmission can occur after each station is polled, rather than only after a complete polling pass.

Auto Poll: For lines polled under the control of the Auto Poll hardware feature, the scheme given above is slightly modified.

If receiving has priority over sending, messages are sent to stations on the line during the invitation delay. However, if no messages have been queued for stations on the line by the time the end of the invitation list is reached, no invitation delay is observed.

If receiving and sending have equal priority, there is no difference between autopolled and other polled lines.

If sending has priority over receiving, outgoing messages are sent over autopolled lines:

1. Each time an EOT is received from a station, indicating that a complete message has been received.
2. Each time the end of the invitation list is reached.

Transmission Priority for Nonswitched Polled Stations Using TCAM's Buffering Feature

The IBM 2740 Model 2 contains a hardware buffer (and a message to the 2740 Model 2 must fit within this buffer); the IBM 2770 on a multipoint line contains two hardware buffers. Messages to these stations fill the buffers at line speed. A message is read from the buffer to the terminal output device at the speed of the output device. This improves line utilization, since the line is occupied with individual stations for a relatively short period of time. If a buffered station is addressed before the buffer has emptied, a negative response is returned and the station must be selected again later. A message to be entered from a buffered station is first entered into the buffer from the input component (at the speed of the input device). When the buffer is filled or the message is entered, the message is transmitted to the CPU at line speed the next time the station is polled.

TCAM sends to an IBM 2740 Model 2 a message at a time and to an IBM 2770 until its buffer space is filled. The 2740 Model 2 accepts messages; thus, a block of data to the 2740 Model 2 must be equivalent to a whole message. To prevent TCAM from trying to send a message to a 2740 Model 2 while the hardware buffer is still emptying the previous message and thus wasting time on the line, TCAM allows the user to specify (in the BFDELAY= operand of the TERMINAL macro) the number of seconds of delay to observe before sending each message after the first to a 2740 Model 2. The time specified should be the average time needed to empty the hardware buffer (the BFDELAY= operand must be specified for the IBM 2770; see BSC device-dependent considerations in the section *Sending Operations* in *Appendix G*). While this interval is in effect, TCAM can be sending messages to other stations on the line, thereby utilizing the line more efficiently.

Thus, when BFDELAY= is specified for the 2740 Model 2 on a multipoint line, messages are sent to stations on the line on a message-by-message basis: the first message is sent; if there are messages queued for other stations on the line, they are sent; subsequent messages are sent as stations become available. For the 2740 Model 2 to become eligible to accept another message, the time interval specified by the BFDELAY= operand of its TERMINAL macro must have elapsed. For information on how to determine the correct interval and restrictions on coding BFDELAY=, see the description of this operand.

When a STOPLINE operator command, a QTAM STOPLN macro, a SYSCLOSE operator command or an MCPCLOSE macro specifying a quick closedown is executed, transmission on a line to stations using TCAM's buffered-terminal support is not stopped until all messages being sent to stations on the line at the time the command or macro is executed have been completely sent and all intervals specified by the BFDELAY= operand of the TERMINAL macros for stations on the line have been observed.

For TCAM's buffering feature to work properly for the IBM 2740 Model 2, queuing by terminal and either equal or send priority must be specified in the TERMINAL and line group DCB macros.

Transmission Priority for Nonswitched Contention Stations

The following can be nonswitched contention stations: the IBM 2740 Basic, the IBM 2780, the IBM 2770, World Trade (WTTA) terminals, and the IBM System 360, System 360 Model 20, and 1130 Computing System. For nonswitched contention stations, either equal or send priority may be specified. The way in which equal priority works is device-dependent, and is explained in *Appendix G.*

Send priority is similar for all these types of stations; if send priority is specified, messages may be entered at the station whenever the line is idle. Whenever a message is queued for sending, TCAM checks to see whether a message is being entered by the station; if so, the computer waits until an EOT control character is received and then sends all messages queued for the station. If no message is being entered, the computer sends all queued messages immediately after checking. After sending all messages, the computer is ready to receive messages from the station. The invitation list for the line may consist of a dummy entry (see the description of the INVLIST macro).

For equal priority for the devices listed above, see *Appendix G: Device Dependent Considerations.*

NOTE: When a BSC device is in contention with the CPU, TCAM defers to the BSC device for control of the line. However, when a start-stop device has a message to enter, and it is in contention with the CPU, the start-stop device loses that message (a Message Handler that includes the SEQUENCE macro can indicate when a message is lost to the system).

Transmission Priority for Switched Stations

For switched stations, CPRI=S must be specified in the line group DCB macro.

The relative order in which messages are sent to and received from a station on a switched line depends upon whether the station is a BSC station or a non-BSC station.

When a non-BSC station calls the computer, once the connection is established the station begins to enter any messages it may have ready for the computer. Before it can accept

messages, the station calling in must identify itself to the computer by means of an origin field, verified by an ORIGIN macro, in a message header. If the station does not identify itself, the computer breaks the line connection upon receiving a negative response to invitation, thereby making the line available for other calls. Once the station identifies itself, it is eligible to accept messages. If any messages were queued for the station at the time it identified itself, the station accepts these messages as soon as possible; no further messages may be entered at the station until the queued messages are sent. Messages are sent by the computer according to the priority scheme outlined in the section *Message Priority and Queuing.* If the destination queue for the station was empty at the time the station identified itself, and once the queue becomes empty during this call, messages are sent to the station as soon as possible after they are placed on the destination queue. That is, whenever a message is completely enqueued on the previously empty destination queue during this call, TCAM checks to see whether a message is being entered by the station; if so, the computer waits until the message has been completely received, and then sends all messages queued for the station. After sending all messages, the computer invites the station to enter messages. When the last incoming message is received and no further messages appear on the destination queue for the station, the computer breaks the line connection, making the line available for new calls.

When the computer calls a non-BSC station, the computer sends all messages queued for the station before the station enters any messages. Messages are sent by the computer according to the priority scheme described in the section *Message Priority and Queuing.* Once all queued messages have been sent, or if the queue was empty, the station begins entering any messages it may have ready. If a message is enqueued for the station after the station begins entering messages, TCAM sends the message as soon as the message currently being received from the station has finished, as described above. When the station indicates that it has no more messages to enter, and no further messages appear on the destination queue, TCAM breaks the line connection, rendering the line available for new calls.

NOTE: See *Appendix G. Device Dependent Considerations* for the transmission priority for:

- 2740 Communications Terminal on a switched line;
- Switched BSC stations;
- Switched TWX stations.

Calls between the Computer and a Switched Station

On a line-group basis, the order in which messages on a switched line are sent and received depends upon whether the computer dials a station or a station dials the computer, and upon when calls are made.

In a TCAM system, a station may call the CPU on a line that is in its own line group and that has a relative line number equal to or greater than the line to which the calling station is assigned. When a station dials the computer, the computer may answer either manually or automatically if it is equipped with the Auto Answer feature.

If these requirements are satisfied, and if the line is not currently connected to another station, a connection is established each time the station dials the number associated with the line. If the line is connected to another station when a station dials its number, the dialing station receives a busy signal and must try again later. Once contact is successfully established between station and computer, message transmission occurs according to the scheme described in the section *Transmission Priority for Switched Stations.*

If the computer is equipped with the Auto Call feature, it may dial switched stations. For the computer to dial a switched station, the station's telephone number must be entered in the DIALNO= operand of its TERMINAL macro (and the computer must be equipped with the Auto Call feature). If CLOCK= is coded for the TERMINAL macro, the computer dials the station only at the time specified by CLOCK=, and this is the only time at which the station may receive messages from the computer.

If CLOCK= is not coded, an attempt is made to call the station whenever a message is completely received and enqueued in the previously empty destination queue for that station. (A destination queue is considered to be "empty" when it contains no completely received, but as yet unsent, messages.)

If the CINTVL= operand of the TERMINAL macro provides an interval, and the station does not call in and is not called during this time, TCAM calls the station at the end of the interval. (When the station calls in or is called during the specified interval, the interval begins again.)

When the first message arrives at the previously empty destination queue for a station (if CLOCK= is not coded), or the time specified by CLOCK= or CINTVL= is reached, the computer attempts to dial the station over the line specified by the RLN= operand of the TERMINAL macro for the station. If this line is currently being used by another station, the computer attempts to place the call over the line whose relative line number is one greater than that specified for this station. If this line is also being used by another station, the computer checks the line whose relative line number is higher by one than that for the line just checked; this procedure is repeated until an available line is found, or until the line having the highest relative line number in this line group is checked and found to be in use. If the line with the highest relative line number in the group is in use, the call is delayed until a line becomes available, at which time it is sent. If more than one waiting call is eligible to be made over a line that has just become available, TCAM decides which call to make according to a priority scheme described below. Once the connection between computer and station is established, transmission occurs in accordance with the scheme described in the section *Transmission Priority for Switched Stations.*

TCAM's calling scheme is designed to take advantage of AT&T's Wide Area Telephone Service (WATS). If WATS is used, care should be taken to arrange the lines in a switched line group to take full advantage of the TCAM calling scheme. Lines should be arranged in a line group according to increasing area of WATS coverage, with the line covering the smallest area being assigned relative line #1, and the line covering the largest area being assigned the highest relative line number in the group. (The way in which lines in a line group are assigned relative line numbers is described in *DD Statement for a Line Group* in the chapter *Defining the MCP Data Sets.*) It is most economical for stations to be assigned to lines whose WATS coverage extend to their area and no farther; in no event should stations be assigned to a line whose coverage does not extend to their location.

When a call cannot be made because all suitable lines in the line group are busy, TCAM queues the request and defers the call until a suitable line is available. If a line becomes available, and if there is more than one call that could be made over the line according to the rules described above (i.e., the line is in the same line group as the line to which the prospective station is assigned, and has a relative line number equal to or greater than that of the line to which the prospective station is assigned), TCAM determines which station will be called first by applying the following principles:

1. A station whose destination queue contains one or more messages having non-zero message priorities is called before a station whose destination queue contains only messages to which no message priority was assigned (i.e., messages having zero priority). A station whose destination queue contains only zero-priority messages is called before a station whose destination queue contains no complete messages.
2. A station having a higher-priority message on its destination queue is called before a station having lower-priority messages on its destination queue. If the highest-priority messages on the queues for two eligible stations are equal in priority (and if this priority is not zero), the time at which the last segments of the high-priority messages were enqueued determines which station is called; the station whose destination queue received the last segment of its highest-priority message first is called first.
3. Among stations having only zero-priority messages on their destination queues, TCAM calls the station whose relative line number is equal to, or closest to but lower than, the relative line number of the available line. Among stations having only zero-priority messages on their destination queues and having the same relative line number, TCAM calls the eligible station whose queue was first to receive a complete message.
4. Among stations whose queues contain no complete messages, TCAM calls the eligible station for which the call has been deferred the longest (this principle is applicable only for stations whose TERMINAL macro specifies CLOCK= or CINTVL=).

Note that a strict WATS priority scheme for deferred calls is observed only among stations whose destination queues contain only messages having zero priority. If relative line #6 becomes available and calls have been deferred for a station assigned to relative line #1 and for a station assigned to relative line #6, and if the queue for the station assigned to relative line #1 contains the highest-priority message, this station will be called

before the other, even though it would be more economical from a WATS standpoint to call the station assigned to relative line #6. (See Principle #2 above.) If the queues for both stations contain only zero-priority messages, a WATS priority scheme will be applied, and the station assigned to relative line #6 will be called first. (See Principle #3 above.)

If the computer dials a station and gets a signal indicating that the station's telephone is already in use, this is treated as an error condition. The station's number will be dialed twice more; if no connection is established in three attempts, TCAM sets the selection error bit in the message error record, and the message is lost unless a REDIRECT or HOLD macro is executed for it in the outmessage subgroup. (The text error bit in the message error record may also be turned on – see the description of this bit in *Appendix B.*). Once the connection between the computer and a switched station is established, transmission occurs according to the scheme described in the section *Transmission Priority for Switched Stations.*

The System Interval

Message flow is vitally affected by the *system interval,* a period of time specified by the INTVAL= operand of the INTRO macro. The INTERVAL operator command tells TCAM to begin the system interval. When this message is received, each multipoint line to polled stations is "frozen" (i.e., polling and addressing cease on it) at the end of its current polling pass, and (if assigned receive priority) after the invitation delay has been observed. When all multipoint lines are inactive, the system interval commences. Lines to switched stations and nonswitched contention lines are left active; stations on such lines may still enter and accept messages. A SYSINTVL operator command may be entered from a contention operator control station to change the duration of the system interval. If this message is entered while a system interval is in effect, it does not change the duration of the current interval, but does change the duration of subsequent intervals.

The system interval is used to minimize unproductive polling, to minimize CPU meter time, and to synchronize polling on the polled lines in the system. In general, if there is no traffic on any line in the TCAM system the OS Supervisor is given control to dispatch the next concurrent job.

Messages entering a TCAM network are read into buffers, which are user-defined areas of main storage used for handling, queuing, and transferring message segments between all lines and queuing media, and between queuing media and application-program work areas. (A message segment is that portion of a message contained in one buffer.) A buffer has two parts, one containing control information (the *buffer prefix*) and the other containing all or part of the message. Buffers must be at least 31 bytes long, and may be no longer than 65535 bytes.

Structure of a Buffer

To provide the best dynamic buffering capability and use of main storage, the TCAM network has one buffer unit pool containing buffer *units* of one size. Buffer units are the basic building blocks from which buffers are constructed.

The size of a *unit* is specified by the KEYLEN= operand of the INTRO macro of an MCP, and the number of units in the pool is equal to the sum of the numbers specified by the LNUNITS= and MSUNITS= operands of INTRO. For internal management purposes, 12 bytes are added by TCAM to the user-specified unit size. Thus, if a user specifies a unit size of 60 bytes (KEYLEN=60), the size of the unit becomes 72. The user should not concern himself with the extra 12 bytes when defining his buffers.

NOTE: If the sum of the number of bytes specified by the KEYLEN= operand plus 12 bytes is not evenly divisible by eight, TCAM adds enough bytes to each unit to make its total length divisible by eight. This is done so that units which are contiguous in main storage always start on a doubleword boundary.

The size of a *buffer* for a line group is specified by the BUFSIZE= operand of the DCB macro defining the line group data set for that group. Each line group may utilize buffers that differ in size from those assigned to other line groups.

By coding the BUFSIZE= operand of the TERMINAL macro, the user may override the buffer size specified in the line group DCB macro on a station-by-station basis, for outgoing messages only.

By linking an appropriate number of units, TCAM constructs buffers containing a number of bytes at least as great as that specified by the BUFSIZE= operand of the DCB macro for a given line group. (The 12 bytes added to each unit by TCAM *should not* be considered in defining BUFSIZE=; the user should consider only the number of bytes he specified in the KEYLEN= operand of the INTRO macro.) For example, if the user specified KEYLEN=60 in the INTRO macro and BUFSIZE=120 in a line group DCB, TCAM links together two units in building buffers for that line group. If, however, KEYLEN=60 and BUFSIZE=100 is coded, TCAM will still link two units, but the last 20 bytes of the second unit cannot be used and main-storage space is wasted. If KEYLEN=60 and BUFSIZE=40 is specified, the last 20 bytes of the first (and only) unit assigned are wasted.

There are two types of logical buffers, header buffers and text buffers. A *header buffer* is a buffer that contains all or any part of a message header. A *text buffer* contains message text only.

A *buffer prefix* is a control area contained within each physical buffer of the system. The user must allow room for the buffer prefix in defining his buffers. TCAM fills in the buffer prefix area with buffer control information.

If only one buffer is used to contain a message, the buffer prefix occupies the first 30 bytes of the buffer. If more than one buffer is used to contain a message, a 30-byte buffer prefix occupies the beginning of the first buffer, and a 23-byte buffer prefix occupies the beginning of each subsequent buffer assigned to the message.

Thus, there are two kinds of control areas associated with buffers. The 12-byte control area associated with each *buffer unit* is assigned automatically by TCAM and need be of no concern to the user when defining buffers. The 30-byte (header) or 23-byte (text) buffer prefix assigned to each *buffer* is of concern to the user, who must allow for this area in defining the size of his units. Each unit must be large enough to contain the larger

prefix plus one byte (31 bytes) and may be no larger than 255 bytes. Obviously, the second and subsequent buffers will contain more bytes of actual message than will the first buffer, since their prefixes are seven bytes shorter than that of the first buffer.

Figure 2 shows how two buffers assigned to a line group would look if the user specified KEYLEN=60 and BUFSIZE=120.

Notice that each buffer is composed of two units linked together, and that the two buffers are also linked together. Each unit is 72 bytes long (the 60 bytes specified by KEYLEN= plus a 12-byte unit control area added by TCAM). In defining BUFSIZE for the line group, only the 60 bytes specified by the user were considered.

Remember that:

- A buffer is composed of one or more buffer units,
- Each buffer *unit* must be at least 31 bytes long (not counting the 12-byte control area added by TCAM) and may be no longer than 255 bytes (not counting the unit control area),
- Each *buffer* must be at least 31 bytes long (minimal size of one unit) and may be no longer than 65535 bytes.

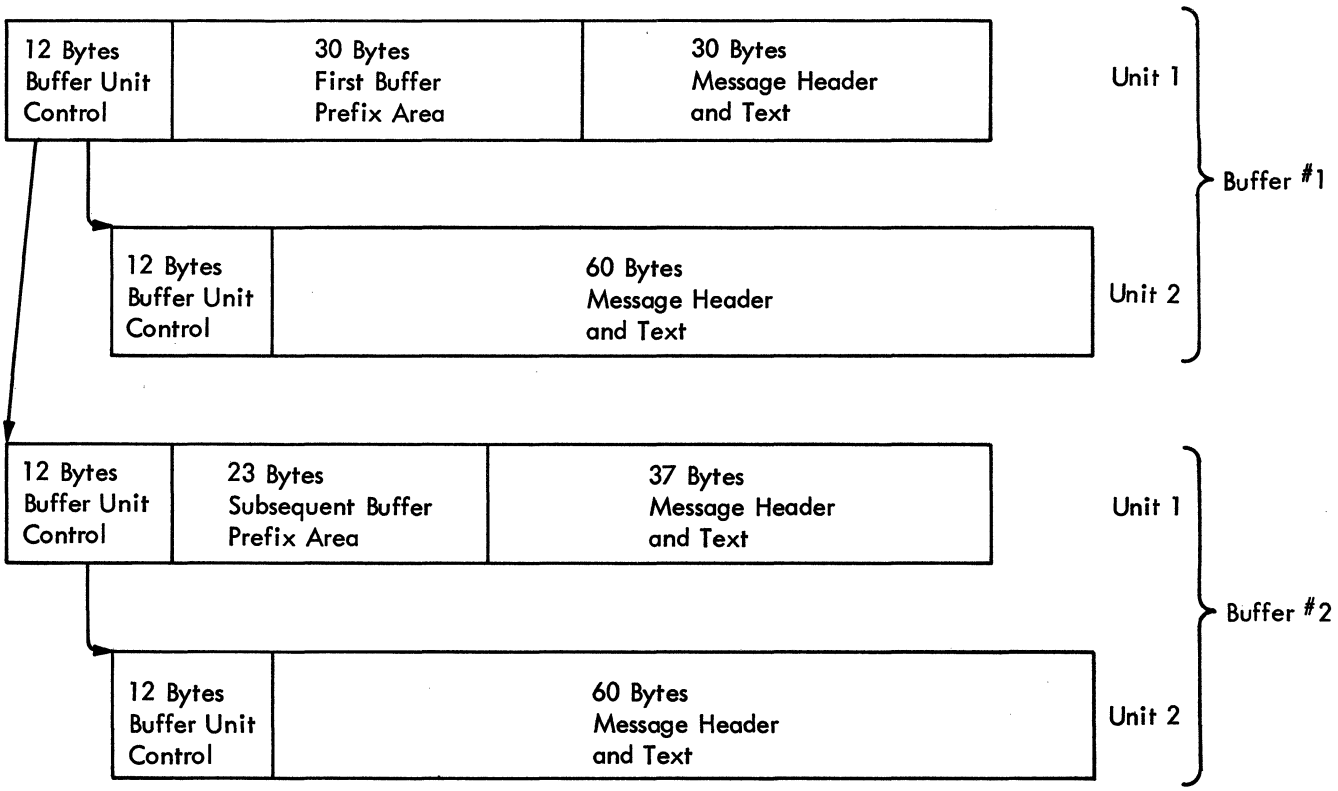


Figure 2. Two Buffers Assigned to a Line Group; KEYLEN=60 and BUFSIZE=120

The Buffer Unit Pool

One buffer unit pool is defined for the Message Control Program. This single pool contains a number of buffer units equal to the sum of the numbers specified by the LNUNITS= and MSUNITS= operands of the INTRO macro. The total number of units in the unit pool must not exceed 65535.

When message traffic is in progress, a unit in the unit pool may be in any one of three states:

1. If a main-storage message-queues data set is specified, some units are assigned to main-storage message queues,
2. Some units are linked to form buffers assigned to line groups or line application programs to handle data transfer,
3. Some units are assigned to an *available-unit queue,* where they remain until linked to form a buffer or until assigned to a message queue.

Figures 3 and 4 show how the units in a unit pool are allocated. Figure 3 illustrates how units are allocated when the user specifies main-storage message queuing with or without backup on reusable or nonreusable disk (see *Defining the MCP Data Sets* for a discussion of main-storage message queuing).

The first block in Figure 3 shows how the unit pool looks just after storage has been allocated for it, when main storage queuing is specified. The pool consists of a number of units equal to the sum of the LNUNITS= and MSUNITS= operands of INTRO. Each unit has a length equal to the number of bytes specified by the KEYLEN= operand of INTRO, plus 12 bytes. All units are assigned to the available-unit queue.

The second block in Figure 3 shows how the pool looks just before selection and invitation begin. A certain number of units have been linked to form buffers, which are assigned to line groups and application programs to handle initial send and receive operations (the number of buffers assigned is specified for line groups by the BUFIN= and BUFOUT= keyword operands of the line-group DCB macro, and for application programs by the BUFIN= and BUFOUT= operands of the PCB macro). All other units are still in the available-unit queue.

The third block in Figure 3 illustrates the situation when normal message traffic is in progress. Some units are in line and application-program buffers; others are in main-storage message queues; the remainder are in the available buffer queue. The arrows represent the normal limits in size of the fraction of the unit pool that can be assigned to line and application-program buffers or to main-storage message queues after selection or invitation has begun. The number of units assigned to main-storage message queues may never exceed the number specified by the MSUNITS= operand of INTRO. The number of units assigned to line and application-program buffers will not ordinarily exceed the number specified by the LNUNITS= operand of INTRO. However, under exceptional conditions (e.g., when main-storage queuing with backup on disk is specified, and there is a peak period of line activity with low main-storage queue activity and high disk activity), the number of units assigned to line and application-program buffers may exceed the number specified by LNUNITS=, if the number of units required is available in the available unit queue.

Figure 4 illustrates how units are allocated when the user has specified disk queuing only for his message queues data set.

The first block in Figure 4 shows how the unit pool looks just after storage has been allocated for it. The pool consists of a number of units equal to that specified by the LNUNITS= operand of INTRO. All units are assigned to the available-unit queue.

The second block in Figure 4 shows how the pool looks just before selection or invitation commence. A certain number of units have been linked to form buffers, which are assigned to line groups and application programs to handle sending and receiving operations. All other units are on the available-unit queue.

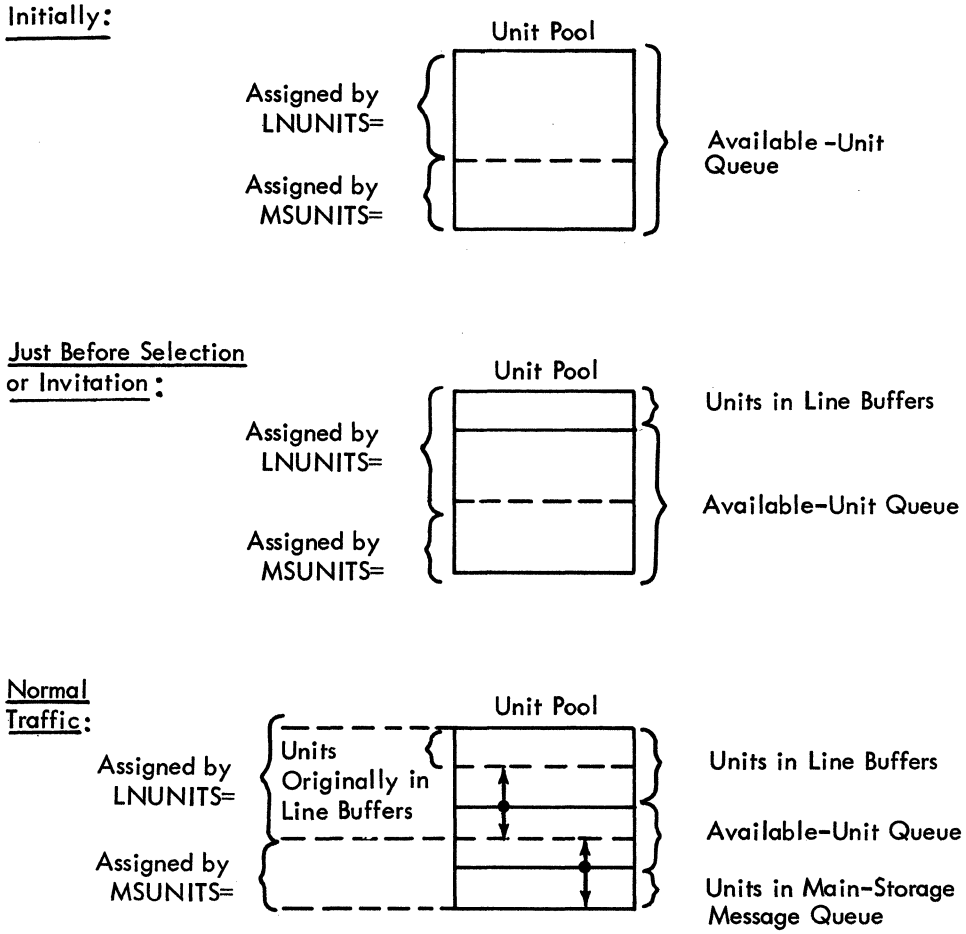


Figure 3. Unit Allocation when Main-storage Queuing (with or without Backup on Disk) is Specified

The third block in Figure 4 illustrates the situation when normal message traffic is in progress. Each unit in the pool is either assigned to a line or application-program buffer or assigned to the available-unit queue. The arrows illustrate the limit in size of the fraction of the unit pool that may be assigned to line buffers after selection or invitation has begun. All units on the available-unit queue may be assigned to line buffers.

NOTE: Buffers are not always available to be assigned to lines; for example, when TCAM does a read operation for a data set residing on disk, a buffer is reserved to hold the record read from the disk.

Buffers assigned to TCAM application programs differ from those assigned to the MCP in the way in which they are defined and in the manner in which they are allocated. For additional information on such buffers, see *Defining Buffers for the Application Program* in the chapter *Writing TCAM-Compatible Application Programs.*

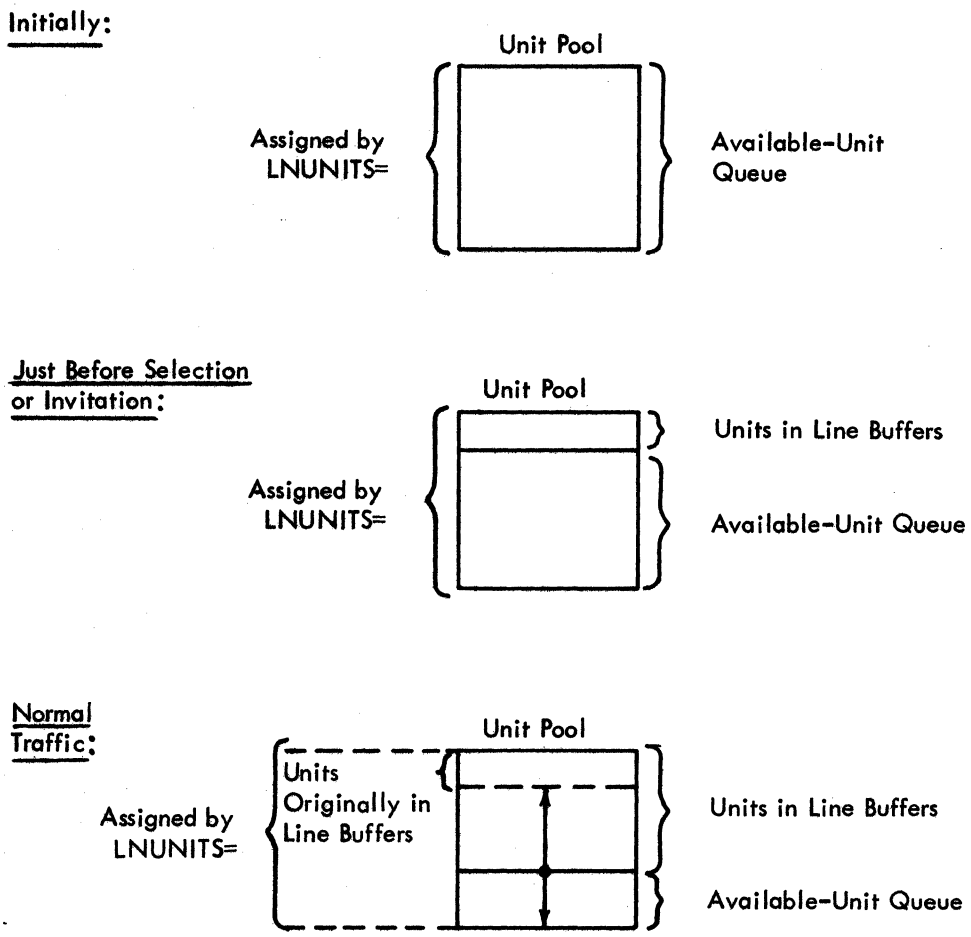


Figure 4. Unit Allocation when Disk-only Queuing is Specified

Buffer Definition Checklist

A checklist of the TCAM macro operands directly involved in MCP buffer definition follows. (A similar checklist for defining application-program buffers is contained in the chapter *Writing TCAM-Compatible Application Programs.*) The macros to which the operands belong are described in detail elsewhere in this publication. The user should first scan the checklist to give himself a general idea of what is involved in defining TCAM buffers, and then read the next section, which contains guidelines for coding many of these operands. Finally, the checklist may be used during actual buffer definition to assure that all applicable operands are coded. For information on maximum and minimum values and defaults, see the operand description for the associated macro.

Macro	*Operand*	*Description of Function and Comments*
INTRO	KEYLEN=integer	Specifies the length in bytes of a buffer unit. The unit as it exists in the unit pool is equal in length to the number of bytes specified by KEYLEN= plus a 12-byte control area added by TCAM. TCAM begins each unit on a doubleword boundary. In order to conserve main-storage space, the following formula can be used as a guideline in determining a value for KEYLEN=: KEYLEN=8x-12 where *x* is any integer between 6 and 33, inclusive. A buffer unit must be large enough to accommodate the larger of: (a) a header prefix (30 bytes) plus the maximum number of reserve characters specified for the first buffer by the RESERVE= operand of any line group DCB macro or PCB macro plus 3 bytes or, (b) a text prefix (23 bytes) plus the maximum number of reserve bytes specified for buffers other than the first by the RESERVE= operand of any line group DCB macro or PCB macro plus 3 bytes.
	LNUNITS=integer	Specifies the number of buffer units in the unit pool that may be used to build line buffers and buffers to handle application-program traffic. The sum of LNUNITS= plus MSUNITS= must not exceed 65535.
Line Group DCB	BUFSIZE=integer	Specifies the size of buffers to be used for all lines in this line group. The size specified here may be overridden on a station basis for outgoing messages by means of the BUFSIZE= operand of the TERMINAL macro. The maximum number of units per buffer is 255.
	[BUFIN=integer]	Specifies the number of buffers assigned initially for receiving operations for each line in the line group.
	[BUFOUT=integer]	Specifies the number of buffers to be assigned initially for sending operations for each line in the line group.
	[BUFMAX=integer]	Specifies the maximum number of buffers allocated to a line at one time. If this operand is omitted, the larger of BUFIN= and BUFOUT= is assumed.
	(PCI=[N|R|A],[N|R|A])	Specifies whether and how program-controlled interruptions (PCI) are to be used for control of dynamic buffer allocation and deallocation. For the meaning of the operands, see the discussion of program-controlled interruptions in *Dynamic and Static Buffer Allocation* in this chapter.
	[RESERVE= (integer 1, [integer 2])]	*integer 1* specifies the number of bytes to be reserved in the first buffer of each incoming message for insertion of data by the DATETIME and SEQUENCE macros. *integer 2* optionally specifies the number of bytes to be reserved in all buffers, except the first, for insertion of characters by the DATETIME macro. See the descriptions of these macros, and the discussion of this operand in the description of the line group DCB macro.
TERMINAL	[BUFSIZE=integer]	Overrides the buffer size specified by the BUFSIZE= operand of the line group DCB macro, but only for buffers containing outgoing messages destined for this station.
LOGTYPE	BUFSIZE=integer	Specifies the size of the buffers to handle messages destined for the logging medium when logging of messages is specified by a LOG macro.

Design Considerations

Management of data buffers for incoming and outgoing messages is an important factor in running a TCAM system at optimal efficiency. There are several factors that a system programmer must consider in weighing the trade-off of time and main storage.

1. The user must specify enough buffer units to assure no loss or undue delay of data.
2. The user must select the size of his buffer units and buffers to accommodate his message.
3. The user must decide whether to use the program-controlled interruption (PCI) feature for control of dynamic buffer allocation and deallocation.
4. The user must determine the number of buffers to be assigned initially to each line in a line group for sending and receiving operations, and the maximum number of buffers to be assigned to each line.

The following lists may aid the system programmer in dealing with the first two of these factors; the other factors are discussed in turn below.

Size of Buffers

Relative Advantages of Larger vs Smaller Buffers

Parameter	Advantages
larger buffers (more units per buffer)	1. Fewer buffers required for a message; consequently overhead required by TCAM to manipulate buffers is decreased. 2. When dynamic allocation of buffers is used, the possibility of losing data because of a delayed PCI is decreased. 3. Number of PCIs required (if PCI is specified) is decreased. 4. Better use is made of the disk accessing method utilized by TCAM (multiple-arm support) because there is a larger number of contiguous records than there would otherwise be. 5. There are fewer queuing operations per quantity of data; this results in a saving of time.
smaller buffers (fewer units per buffer)	1. Units in smaller buffers tend to be returned to the available-unit queue more rapidly than would be units in larger buffers (since it takes less time to empty and fill a smaller than a larger buffer). Since units in smaller buffers are available for reuse sooner than equivalent units in larger buffers would be, a smaller unit pool is possible when smaller buffers are used. 2. When smaller buffers are used, TCAM's work load is broken into smaller pieces; this results in a more equitable allocation of processing time among message segments in main storage.

Number of Units

Relative Advantages of Having Many vs Few Units in the Pool

Parameter	Advantages
more units in system	1. Likelihood of losing message data coming in over a line is decreased. 2. Outgoing messages are less likely to be delayed as a result of waiting for a buffer.
fewer units in system	1. Main storage is utilized more efficiently. Since the number of units in the free unit pool is not excessive, main storage is saved.

Size of Units

Relative Advantages of Larger vs Smaller Units

Larger units

1. Disk space is utilized more efficiently, since there are fewer interrecord gaps.
2. Proportion of area available for text to area containing management information is relatively large.
3. Since more data is transmitted per CCW on line and disk, channel activity is relatively light; this results in a saving of channel fetch time and CPU time.
4. Fewer channel program blocks (CPBs) are needed for transferring the same amount of data to and from disk; this results in a saving of storage space and time (since there is less queuing of CPBs).

Smaller units

1. Duplicate headers (used for multiple routing of messages) take up relatively little room.
2. User can specify a relatively large range of buffer sizes without wasting space in main storage and on disk.
3. Allocation of buffers can be more dynamic with smaller units, since smaller units are passed around the TCAM system more rapidly than larger units.

Dynamic and Static Buffer Allocation

When the PCI= operand of the DCB for a line group is coded to permit program-controlled interruptions, a PCI may occur during the filling of the first and each subsequent buffer assigned to a line group. When this interruption is received, control is given to a TCAM PCI routine.

If PCI=A is coded, when the first interruption occurs a number of buffers equal to the difference between the maximum number assigned to a line group (specified by the BUFMAX= operand of the DCB) and the number initially assigned to the line group (specified by the BUFIN= operand of the line group DCB for a receiving operation and by the BUFOUT= operand for a sending operation) is assigned as soon as possible to the line group. On subsequent PCIs, the buffer immediately preceding the one being filled or emptied is deallocated (for a sending operation, the buffer units are returned to the available unit queue; for a receiving operation, the buffer is sent to the Message Handler for that line group) and a new buffer is requested to keep the number of buffers assigned to the line group equal to that specified by BUFMAX=.

When PCI=R is coded, the previous buffer is deallocated when the second and subsequent PCIs occur, but no requests are made for additional buffers. If program-controlled interruptions are *not* permitted (PCI=N), or if only deallocation is specified (PCI=R), then the number of buffers assigned initially must be sufficient to handle the entire transmission. If PCI=N is specified, no deallocation of buffers occurs until the transmission is completed, or, if EOB checking is specified in the STARTMH macro, until an EOB control character is received.

Advantages:

- When PCI=A is coded, fewer buffers need be assigned initially to a line, since dynamic allocation brings the number of buffers assigned up to the value specified by BUFMAX= and maintains this number if possible.
- When PCI=A is coded and a negative response to invitation occurs, only the number of buffers assigned initially, rather than the maximum number assigned to the line, have been fruitlessly allocated.
- When PCI= is specified as A or R, buffers are continuously being deallocated; the free-unit pool is therefore continuously being replenished and a smaller unit pool is required.

Disadvantages:

- Dynamic allocation and deallocation of buffers takes processing time.

NOTE: In order for dynamic allocation to work properly for BSC lines, the BUFMAX= operand of the line group DCB macro must specify a value that is at least two greater than that specified by the larger of either the BUFIN= or the BUFOUT= operand of the line group DCB macro, unless the lengths of all messages (including prefixes) is less than or equal to the total length of the number of buffers specified by BUFMAX=. For start-stop lines using dynamic allocation, a specification of BUFIN=2, BUFMAX=2 may cause inefficient dynamic allocation.

Initial and Maximum Number of Buffers per Line

The number of buffers that should be assigned initially to each line in the line group (by the BUFIN= and BUFOUT= operands of the line group DCB macro) depends upon the following factors:

- terminal type;
- terminal speed;
- line speed;
- whether dynamic allocation of buffers is specified.

The number of buffers to be assigned initially varies directly with the speed of the line and the terminal; the faster the data is transmitted, the higher the initial assignment should be.

The maximum number of buffers assigned to a line in the group (by the BUFMAX= operand of the line group DCB macro) also depends upon the line and terminal speed. For a system using dynamic allocation of buffers, allowance should be made for the fact that program-controlled interruptions might not be accepted by the CPU in time for buffer replenishment to be effective for any particular buffer. For high-speed BSC lines, dynamic allocation may not be totally effective; that is, there may not be a one-to-one correspondence of replacement buffers to replaced buffers. If this happens consistently, incoming data may be lost and bit 6 turned on in the message error record. The higher the line speed, the greater the disparity may become. When dynamic allocation is not used by the system, BUFMAX= is ignored.

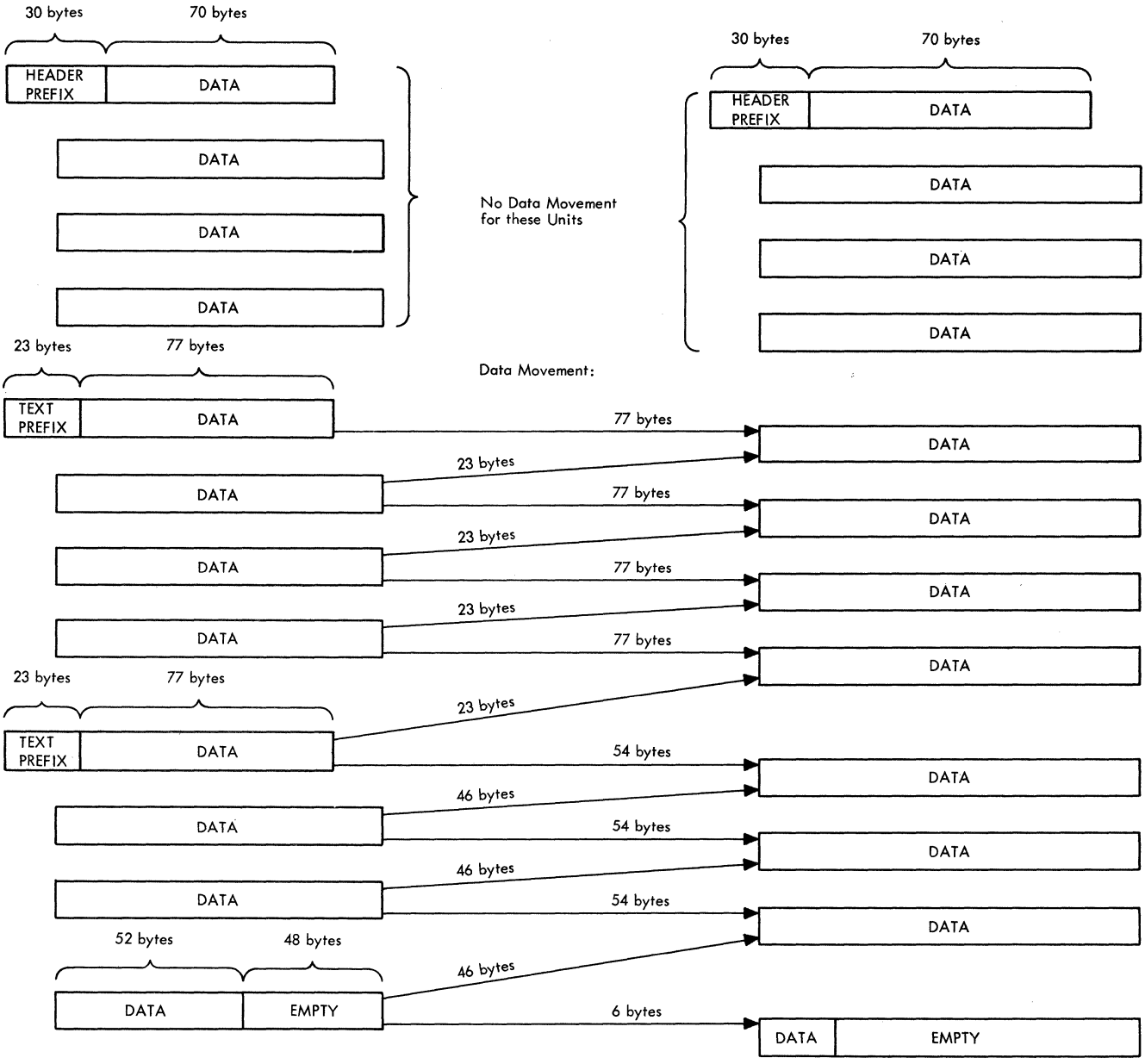
NOTE: The buffers assigned to each line in a line group by the BUFIN= operand of the line group DCB macro and the buffers assigned to each line by the BUFOUT= operand of the line group DCB macro are never assigned to the same line at the same time. The buffers specified by BUFIN= are assigned to a line just before a station on that line is invited by TCAM to enter a message, while the buffers specified by BUFOUT= are assigned immediately before a station on the line is selected to receive a message. Hence, when the user is deciding how many units to define to handle initial line operations, he need consider only the larger of the values specified by BUFOUT= and BUFIN= for each line in a line group, and not the sum of the two values.

Other Buffer Design Considerations

- If the buffer size (as specified by the BUFSIZE= operand of the line group DCB, TERMINAL, LOGTYPE or PCB macro, or the BUFL= operand of the input or output DCB) is not a multiple of the effective unit size (as specified by the KEYLEN= operand of the INTRO macro), buffer space is wasted. For example, if the INTRO macro specifies KEYLEN=36 and the DCB macro for a line group specifies BUFSIZE=100, 108 bytes (i.e., 36 X 3) are assigned to the buffer, but only 100 bytes are available for prefix and message data. Thus, 8 bytes are wasted for each such buffer.
- If disk queuing is used, an attempt should be made to ensure that the buffer size specified by the source of a message is equal to the buffer size specified by the destination. The source of a message may be either a station or an application program. If it is a station, that station's line group DCB macro determines the buffer size of messages that it may enter; if it is an application program, a PCB macro determines buffer size. The destination for a message also may be either a station or an application program. Buffer size for an accepting station is determined either by that station's line group DCB macro or a TERMINAL macro (if the buffer size is specified on the TERMINAL macro, this value overrides the value specified on the line group DCB). A PCB macro determines buffer size for an application program that is the destination for a message. When the buffer sizes specified for the origin and the destination of a message are different, data movement occurs because of the necessity of adding or deleting prefixes when the message is placed in the buffers for the destination. (The message is

queued on disk with its old prefixes; when it is removed from a queue and placed in buffers of a different size, prefixes must be added or removed and message data must consequently be shifted.) Movement of data takes time. Figure 5 illustrates a situation in which 706 bytes of a 1076-byte message must be moved because of a difference in origin and destination buffer size.

Buffering and queuing are closely related concepts; the discussions of main-storage and disk queuing in the chapter *Defining Data Sets* should be read in conjunction with the present chapter.



Relevant Macro Operands

MACRO	OPERAND
INTRO	KEYLEN=100
Line Group DCB for Incoming Line	BUFSIZE=400
Line Group DCB for Outgoing Line	BUFSIZE=1200, BUFOUT=1

message length=1076 bytes.

Figure 5. 706-byte Data Movement Resulting from Size Disparity between Input and Output Buffers.

The Message Control Program may refer to four types of data sets. Two of these are required for every MCP, while two are optional. Required data sets are:

- The line group data set and
- The message queues data set.

Optional data sets are:

- The checkpoint data set, if the checkpoint facility is desired,
- The log data set, if the logging function is desired.

Log data sets are not TCAM data sets; they are discussed briefly below. Other data sets are needed if there are any application programs; these data sets are described in the chapter dealing with TCAM support for application programs.

With one exception (a message queues data set in main storage with no disk backup), TCAM data sets are defined by the user by DCB macro instructions. The total number of TCAM MCP data sets may not exceed 255.

Line Group Data Sets

A line group data set consists of the lines in a line group over which messages are transmitted to and from the Central Processing Unit. The user must specify one line group DCB macro instruction for each line group in the system.

Characteristics of a Line Group

A line group may consist of from one to 255 lines. (The size of a line group is also limited by the fact that the INVLIST= operand of the line group DCB macro can be no longer than 255 characters, including commas; thus, if each of 255 lines has an invitation list associated with it, the lines cannot all be accommodated within the same line group.) All lines in the group must have the following common characteristics:

- Either all lines in the group are switched or all are nonswitched.
- Either all lines in the group use start-stop transmission or all use binary synchronous transmission.
- All lines are associated with stations having the same device characteristics.
- All lines are preassigned the same number of buffers to handle the initial segments of incoming messages.
- All lines use the same invitation delay.
- All lines use the same Message Handler.
- No line in the group is a member of another group.

Creating a Line Group Data Set

A line group data set is defined by a line group DCB macro instruction, which creates a data control block for the line group. Parameters based on the keyword operands specified in the macro are included in the data control block.

Operands of the line group DCB macro enable one to specify functions concerned with buffering (BUFOUT, BUFIN, BUFSIZE, BUFMAX, PCI), polling (INTVL, CPRI, INVLIST), and message translation (TRANS) on a line-group basis. These operands are described in detail in the next section. Various aspects of polling and translation are discussed in the chapter *Defining Terminal and Line Control Areas*, while the chapter *Defining Buffers* includes a discussion of how best to code the DCB operands concerned with buffering.

Line Group DCB Macro Instruction

The line group DCB Macro:

- Defines a line group data set;
- Must be issued for each line group in the TCAM system;
- Identifies the Message Handler for the lines in this line group;
- Identifies the invitation lists assigned to the lines in this group;
- Specifies the invitation delay;
- Indicates transmission priority for stations on lines in this group;
- Specifies the number of buffers assigned initially to lines in this group for sending and receiving operations;
- Specifies when buffers servicing lines in this group are to be allocated and deallocated;

- Specifies the buffer size for buffers servicing lines in this group;
- Specifies the maximum number of buffers assigned to a line at one time;
- Specifies the number of bytes to be reserved for insertion of certain data into buffers;
- Specifies the translation tables to be used in translating incoming and outgoing messages.

The line group DCB macro defines a data control block for a line group data set. Parameters based on the keyword operands specified in the macro are included in the data control block. One line group DCB macro must be issued for each line group in the TCAM system. The macro generates no executable code.

The line group DCB macro has the following format:

Name	*Operation*	*Operands*
linedcb	DCB	keyword operands

linedcb

Function: Specifies the name for the macro instruction and also for the data control block generated by the expansion of the macro.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

keyword operands

Function: Are the operands that can be specified. They are described in the list of operands that follows.
Notes: The operands may be specified in any order and are separated by commas with no intervening blanks. See *Appendix A* for a description of the format and symbols that define macro operands. When a parameter can be provided by an alternate source, a symbol appears in the Alternate Source description for the operand associated with that parameter. When there is not an alternate source (that is, the parameter must be specified by the operand), the Alternate Source descriptor states *none*. The symbols have the following meanings:

Symbol	*Explanation*
DD	The value of the operand can be provided at execution time by data definition (DD) card for the data set. If a value is provided by a DD statement, the macro operand must be either omitted or coded with a zero value (if the operand is omitted, a zero value is supplied by TCAM).
PP	The value of the operand can be provided by the user's problem program any time before the data control block exit at open time.
OE	The value of the operand can be provided by the problem program any time up to and including the data control block exit at open time.

NOTE 1: If DD is specified, OE or PP may also be used. If OE is specified, PP may also be used. For information on providing parameters via DD, see the section *DD Statements for a Line Group.* For information on providing parameters via OE or PP, see Note 1 following the description of DD, OE, and PP in *Message Queues DCB Macro Instruction* in this chapter.

NOTE 2: The formats of macro illustrations, the symbols used in them, and rules for the interpretation of operand descriptions, are all provided in *Appendix A.*

DSORG=TX

Alternate Source: None.
Function: Identifies the data set organization as that for a line group.
Default: None. This operand is required.
Format: DSORG=TX

MACRF=(G,P)

Alternate Source: None.
Function: Specifies that access to the line group is gained with GET and PUT macro instructions.
Default: None. This operand is required.
Format: MACRF=(G,P)

INTVL={integer / **0**}

Alternate Source: PP, OE, DD.
Function: Specifies the number of seconds of invitation delay (that is, the number of seconds of intentional delay between passes through an invitation list).
Default: INTVL=0
Format: Unframed decimal integer.
Maximum: 255.
Notes: After all the stations in an invitation list for a given line have been invited to enter a message, a delay equal to the number of seconds specified in this operand occurs before invitation is restarted at the beginning of the list. An appreciation of the value of the invitation delay may be gained by reading *Transmission Priority* in the chapter *Defining Terminal and Line Control Areas.*

CPRI={**R** / **E** / **S**}

Alternate Source: PP, OE, DD.
Function: Specifies the relative transmission priority assigned to the lines in this line group.
Default: None. This operand is required.
Format: R, E or S.
Notes: R specifies that CPU receiving has priority over CPU sending. E specifies that receiving and sending have equal priority. S specifies that CPU sending has priority over CPU receiving.

See *Transmission Priority* in the chapter *Defining Terminal and Line Control Areas* for a discussion of the effect of transmission priority on the times when messages can be sent on the line.

DDNAME=ddname

Alternate Source: PP.
Function: Specifies the name that appears in the DD statement associated with the data control block.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.

EXLST=address

Alternate Source: PP.
Function: Specifies the address of the problem program exit list.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: This list must be provided if data control block or user ABEND exits are required. The list must start on a fullword boundary. Its format and contents are shown in the *OS Data Management Services* publication. User ABEND exits are described in the last section of this chapter.

BUFIN={integer / **1**}

Alternate Source: PP, OE, DD.
Function: Specifies the number of buffers assigned initially for receiving operations for each line in the line group.
Default: BUFIN=1. This default is supplied at open time, rather than at assembly time.
Format: Unframed non-zero decimal integer.
Maximum: 15.
Notes: These buffers are assigned just before a station is permitted to enter a message. BUFIN=, BUFOUT= and BUFMAX= must all be specified from the same source.

For more information on initial assignments of buffers, see the chapter *Defining Buffers.*

BUFOUT={integer / **2**}

Alternate Source: PP, OE, DD.
Function: Specifies the number of buffers assigned initially for sending operations for each line in the line group.
Default: BUFOUT=2. This default is supplied at open time, rather than at assembly time.
Format: An unframed decimal integer greater than 1.
Maximum: 15.
Notes: BUFIN=, BUFOUT= and BUFMAX= must all be specified from the same source.

BUFMAX={integer / 2}

Alternate Source: PP, OE, DD.
Function: Specifies the maximum number of buffers used for data transfer for each line in this line group.
Default: BUFMAX=2. This default is supplied at open time, rather than at assembly time.
Format: Unframed decimal integer greater than 1.
Maximum: 15.
Notes: Must be no less than the larger of the numbers specified by BUFIN= and BUFOUT=.

BUFIN=, BUFOUT= and BUFMAX= must all be specified from the same source.

BUFSIZE=integer

Alternate Source: PP, OE, DD.
Function: Specifies the buffer size in bytes used for all lines in this line group.
Default: None. Specification optional.
Format: Unframed decimal integer greater than 35.
Maximum: 65535.
Notes: The size specified here may be overridden for outgoing messages on a station basis by the BUFSIZE= operand of the TERMINAL macro. If the buffer size is not an even multiple of the buffer unit size specified by the KEYLEN= operand of the INTRO macro, storage space is wasted. The maximum number of units per buffer is 255.

INVLIST= (listname [,{A / B}],[{A / B}] ,,...)

Alternate Source: None.
Function: Specifies the names of the invitation lists for the lines of the line group.
Default: None. This operand is required.
Format: Each listname is the name specified for the INVLIST macro used to define the list for that line. Listnames are specified according to the ascending relative line numbers of the lines in the group. The maximum total length of the data coded for this operand is 255 bytes. A and B are coded as shown.
Notes: For information on relative line number, see *DD Statements for a Line Group* in this chapter.

There must be one invitation list name in the sublist for each line in the line group. If a line is used for output only, a dummy invitation list name with no entries is specified. Any number of output only lines may refer to the same name. No list other than a dummy invitation list may be named by more than one line.

For information on invitation lists, see the *Invitation* section in *Defining Terminal and Line Control Areas.*

The two sets of A/B suboperands are meaningful only for lines attached to a channel through an IBM 2701 Transmission Control Unit, in which case they have the following meanings:

The first A specifies that communications are to be through the 2701 Data Adapter Unit's Dual Communication Interface A.

The first B specifies that communications are to be through the 2701's Dual Communication Interface B. This parameter is not coded if this feature is not present on the 2701.

The second A specifies that transmission will be in Code A for 2701 Data Adapter Unit Dual Code Feature.

The second B specifies that transmission will be in Code B for 2701 Dual Code Feature. This parameter is not coded if this feature is not present on the 2701.

A is the default value for both sets of suboperands.

NOTE: If either or both of the A/B suboperands are omitted, the commas that precede them must still be coded. For example, if the names of the invitation lists for the lines in this line group are LIST1, LIST2, and LIST3, and if the A/B suboperands are to be omitted from this operand, the operand might be coded as follows:

```
,INVLIST=(LIST1,,,LIST2,,,LIST3)
```

MH=mhname

Alternate Source: None.
Function: Specifies the name of the Message Handler for the line group represented by this DCB macro.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols, and must be the same as the name specified in the name field of a STARTMH macro.

PCI= ({N / R / A}, {N / R / A})

Alternate Source: PP, OE, DD.
Function: Specifies if and how a program-controlled interruption (PCI) is to be used for control of buffer allocation and deallocation.
Default: PCI=(A,A). This default is supplied at open time, rather than at assembly time.
Format: Framing parentheses required. N, R and A coded as shown.
Notes: The suboperands apply to receiving and sending operations respectively. N specifies that no PCIs are taken during filling (on receive operations) or emptying (on send operations) of buffers. Buffers are deallocated at the end of transmission.

R specifies that after the first buffer is filled (on receive operations) or emptied (on send operations), a PCI occurs during the filling or emptying of each succeeding buffer. The completed buffer is deallocated, but no new buffer is allocated to take its place.

A specifies that after the first buffer is filled (on receive operations) or emptied (on send operations), a PCI occurs during the filling or emptying of the next buffer. The first buffer is deallocated. A buffer is allocated in place of the deallocated buffer.

The program-controlled interruption is more thoroughly described in the chapter *Defining Buffers.*

RESERVE=({integer1 / 0}[,{integer2 / 0}])

Alternate Source: PP, OE, DD.
Function: integer1 specifies the number of bytes reserved in the buffer receiving the first incoming segment of each message entered on a line in this line group for insertion of characters by the DATETIME and SEQUENCE macros. *integer2* specifies the number of bytes reserved in all buffers except the first, for insertion of characters by the DATETIME macro.
Default: RESERVE=(0,0)
Format: Unframed decimal integers.
Maximum: For each, 255.
Notes: integer2 is relevant only in a multiple-buffer header situation when DATETIME is to insert data in a portion of the header not in the first buffer (see the description of DATETIME for an example showing when it might be desirable to execute DATETIME on a portion of the header not located in the first segment).

Data may be inserted in either an incoming or an outgoing message header, but space must be reserved in the incoming header. On the outgoing side, reserved space is retained for the first buffer only; thus, a DATETIME or SEQUENCE macro, if specified in an out-header subgroup, must operate on the first segment of the message. No space need be reserved for data inserted by means of a MSGEDIT macro.

The *Scan Pointer* section of the chapter *Designing a Message Handler* describes how TCAM handles reserve bytes.

Each integer must be at least one less than the value specified in the BUFSIZE= operand. Each buffer containing header data should be large enough to accommodate the segment itself plus any data that may be inserted by means of DATETIME and SEQUENCE macros. If a buffer containing header data does not have a sufficient number of bytes reserved in it to accommodate data inserted by a DATETIME or SEQUENCE macro, the macro does not execute, and control passes to the next instruction in the MH. Unused reserve bytes are removed from an outgoing message segment when it is sent to its destination.

TRANS={table / EBCD}

Alternate Source: None.
Function: Specifies the translation table for this line group.
Default: TRANS=EBCD

Format: Either the name of a user-defined table conforming to the rules for assembler language symbols, or one of the following four-byte symbols:

1030	1030 transmission code
1050	1050 transmission code
105F	Folded 1050 transmission code
1060	1060 transmission code
2260	2260 Remote transmission code
2265	2265 transmission code
2740	2740 transmission code
274F	Folded 2740 transmission code
BC41	2741 BCD code
EB41	2741 EBCDIC code
CR41	2741 Correspondence code
ITA2	World Trade terminals transmission code
ZSC3	World Trade terminals transmission code
TTYA	83B3, 115A transmission code
TTYB	33/35 parity transmission code
TTYC	33/35 non-parity transmission code
6BIT	2780 6-Bit transmission code
ASCI	2780, 360 CPUs, Model 20 ASCII transmission code
EBCD	2770, 1130, 2260 Local, 2780, 360 CPUs, Model 20 EBCDIC transmission code.

Notes: Specification of a user-defined table is described in *Message Translation* in the chapter *Designing the Message Handler.*

Translation is from transmission code to EBCDIC for incoming messages and from EBCDIC to transmission code for outgoing messages. For incoming translation to occur, a CODE macro must be executed in the incoming group handling the message. For outgoing translation to occur, CODE must be executed in the outgoing group handling the message. If this operand is omitted, no translation is performed.

The table specified for translation can be changed for messages from a particular line or station by the CODE macro and the use of path switches.

TRANS=EBCD should be coded for lines to stations using EBCDIC line code, if any of the stations may enter operator commands (TCAM uses the CODE macro to check for operator commands).

For more information on the symbols, and on translation in the TCAM system, see *Message Translation* and the description of the CODE macro in the chapter *Designing a Message Handler.*

SCT=table

Alternate Source: None.
Function: Specifies the name of the special characters table (SCT) used for this line group.
Default: The table specified for the TRANS= operand is assumed for the SCT= operand if this operand is omitted.
Format: One of the four-byte codes permissible for the TRANS= operand.
Notes: If a user-specified table is coded for the TRANS= operand, the SCT= operand must be coded. The SCT is an internal TCAM table containing certain device-specific line-control characters needed by TCAM whether or not line-control characters are left in incoming messages. TCAM makes no provision for the user to specify his own special characters table.

The contents and layout of the SCT are described in the *TCAM PLM.*

DD Statements for a Line Group

At least one DD statement must be issued for each line group data set. Either of two schemes may be followed in issuing DD statements for line groups:

1. If at system generation time a UNITNAME macro is issued to specify the lines in a line group and to assign to them a single name, then a single DD statement may be issued for this line group at MCP execution time. This DD statement would have the format

```
//ddname DD UNIT=(name,n)
```

where *ddname* is the name specified by the DDNAME= operand of the DCB macro for the line group, *name* is the name assigned to this group of lines by the NAME= operand of the UNITNAME macro, and *n* is the number of lines to be allocated from among the lines whose hardware addresses are coded in the UNIT= operand of UNITNAME.

Example:
At system generation time, the following UNITNAME macro was issued to define a group of lines:

```
UNITNAME UNIT= (021,022,024,025)
NAME=GROUPONE
```

(The four numbers in the UNIT= operand are the hardware addresses of four lines, and are assigned to the lines by IODEVICE macros at system generation time.) At execution time for the Message Control Program, the following DD statement might be issued for this line group:

```
//ddname DD UNIT=(GROUPONE,4)
```

In this case, the line group data set would consist of the four lines defined by the UNITNAME macro. Relative line numbers are assigned to the lines in the same order as they appear in the UNIT= operand of the UNITNAME macro. If the UNIT parameter of the DD statement were coded UNIT=(GROUPONE,2), the line group data set would consist of only the first two lines specified in the UNIT= operand of the UNITNAME macro.

2. A DD statement may be issued for each line in a line group; these DD statements are concatenated as follows (assume that the line group consists of three lines):

```
//ddname    DD UNIT=address
//          DD UNIT=address
//          DD UNIT=address
```

where *ddname* is the name specified by the DDNAME= operand of the DCB macro for the line group, and *address* is the hardware address of the line, as assigned at system generation time by means of an IODEVICE macro. Note that DD statements for all lines in a line group are listed under a single ddname. When this scheme is used, the order in which the DD statements for a line group are arranged determines the relative line numbers specified in TERMINAL macros; i.e., the first line specified is relative line number one, the second line specified is relative line number two, etc., (see the discussion of the TERMINAL macro in the chapter *Defining Terminal and Line Control Areas*).

NOTE: The type of stations on lines in the line group for which the DD statement is issued must be the same as the type specified by the IODEVICE macro that defines the line at system generation time. Be sure that the line you specify in the UNIT= parameter of your DD statement can handle the stations assigned to that line via TERMINAL macros. Otherwise, the data set will not open.

Certain of the line group DCB macro operands may be omitted from the DCB macro and be specified at MCP program execution time by coding them as subparameters in the DCB parameter of the first DD statement for a line group. The way in which the DCB parameter would be coded to specify any one of these DCB macro operands is as follows (BUFIN=, BUFOUT= and BUFMAX= must all be specified from the same source):

```
DCB=(BUFIN=integer,BUFOUT=integer,BUFMAX=integer)
DCB=(INTVL=integer)
DCB=(CPRI={R|E|S})
DCB=(RESERVE=(integer[,integer]))
DCB=(PCI=({N|R|A},{N|R|A}))
DCB=(BUFSIZE=integer)
```

These subparameters are described in the discussion of the line group DCB macro. More than one DCB operand may be specified in this manner.

If the above DCB operands are still zero after OPEN, the following defaults are used:

```
BUFIN=1
BUFOUT=2
BUFMAX=2
CPRI=S
RESERVE=0
PCI=(N,N)
BUFSIZE=value of KEYLEN= on INTRO.
```

Example:
The following DD statements define a line group consisting of three lines. The PCI= operand was not specified in the line group DCB macro, but is being specified at program execution time on the DD statement.

```
//ddname DD UNIT=024,DCB=(PCI=(R,R))
//         DD UNIT=022
//         DD UNIT=025
```

In this example, the line whose address is 024 is assigned relative line number 1, the line whose address is 022 is assigned relative line number 2, and the line whose address is 025 is assigned relative line number 3.

Message Queues Data Sets

In a TCAM system, messages entered by remote stations are queued by destination. A destination may be a station on a line or an application program. Because each incoming message is placed on a queue for its destination rather than being sent to the destination immediately, overlap of line usage in I/O operations is possible. Messages having a common destination may be received simultaneously from more than one source, and the destination itself may also be entering or accepting a message.

Destination queues for each destination (line, terminal, or application program), and a queue for each logging medium used (for message logging), are located in one or more message queues data sets, which may reside either in main storage or on a direct-access storage device. Messages may be queued

- On reusable disk;
- On nonreusable disk;
- In main storage only;
- In main storage with backup on reusable disk;
- In main storage with backup on nonreusable disk.

Although there are five queuing techniques, a maximum of three message queues data sets need be defined; one on reusable disk, one on nonreusable disk, and one in main storage.

In the following discussion we shall first explain each of the five message queuing techniques, giving their relative advantages and disadvantages, and then describe how each may be implemented.

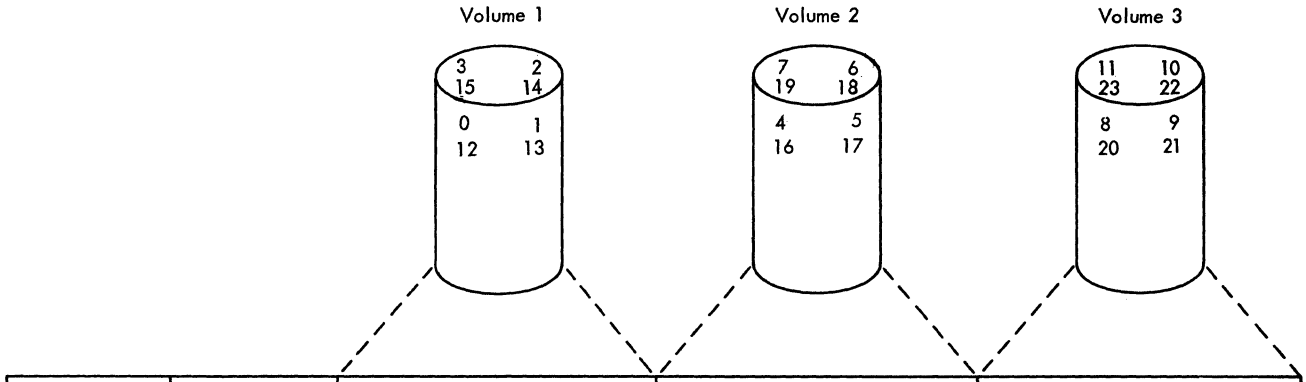
Messages may be queued by destination line or by destination terminal; this topic is discussed in the *Message Priority and Queuing* section of the *Defining Terminal and Line Control Areas* chapter.

Disk Queuing

TCAM supports secondary-storage message queuing on the IBM 2311 Disk Storage Drive, and on the IBM 2314 Direct Access Storage Facility.

The objective of TCAM's secondary-storage queuing scheme is to optimize channel and disk performance. Rotational delay time is minimized through use of sequential disk records wherever possible. The user may specify more than one DASD volume for a data set; if he does, TCAM assigns relative record addresses across volumes, so that the next relative record address after that of the last record on a track is on another volume. Figure 6 illustrates this relative-record addressing scheme, which facilitates efficient multiple-arm support. TCAM's multiple-arm support (described below) permits overlap of seek time on multiple volumes and overlap of channel operations on multiple channels.

Seek time is further optimized by minimizing arm movement.



Cylinder	Track	Relative Record Numbers	Relative Record Numbers	Relative Record Numbers
0	0	0 1 2 3	4 5 6 7	8 9 10 11
	1	12 13 14 15	16 17 18 19	20 21 22 23
	2	24 — 27	28 — 31	32 — 35
	3	36 — 39	40 — 43	44 — 47
	4	48 — 51	52 — 55	56 — 59
	5	60 — 63	64 — 67	68 — 71
	6	72 — 75	76 — 79	80 — 83
	7	84 — 87	88 — 91	92 — 95
	8	96 — 99	100 — 103	104 — 107
	9	108 — 111	112 — 115	116 — 119
1	0	120 — 123	124 — 127	128 — 131
	1	132 ...		

Figure 6. Relative Record Numbers of Disk Message Queues Data Set Assigned Across Three Volumes

Advantages and Disadvantages of Disk Queuing

Locating destination queues in a message queues data set residing on a disk rather than in a data set residing in main storage with no disk backup results in certain advantages:

1. Locating queues on disk rather than in main storage results in more main storage being available to the user.
2. With disk queuing, messages being sent to a station that is temporarily inoperative may be intercepted by a HOLD macro issued in the Message Handler, and sent out at a later time. The interception facility is not available for destinations whose queues are located in a main-storage data set having no disk backup.
3. By issuing a POINT macro in conjunction with a GET or READ macro in an application program, the user may retrieve from its destination queue the original copy of a message that has already been successfully transmitted to a destination station or sent to an application program. This retrieval capability (discussed in the *Message Retrieval* section of the chapter *Writing TCAM-Compatible Application Programs*) might be used to permit a message that was successfully sent to a terminal but lost at the terminal (due, perhaps, to a tape breakage) to be retransmitted. Messages may not be retrieved from main-storage queues.
4. When disk queuing is used, it is possible to take advantage of the TCAM checkpoint/restart facility, which is described in the chapter *Using TCAM Service Facilities.* Main-storage queues cannot be checkpointed; unless disk backup is provided, the data in such queues is lost when the TCAM system closes down or fails.

Locating message queues in a data set on disk rather than in a main-storage data set also has certain disadvantages:

1. Disk queuing is slower than main-storage queuing; that is, a message that is queued on disk takes longer to reach its destination than a message that is queued in main storage, all other things being equal.
2. Disk queuing ties up disk space and disk channels that otherwise could be used by other jobs (for example, by a batch-processing job) in a computing system not dedicated to TCAM.

Main-storage queuing with disk backup (discussed below) preserves most of the advantages of disk queuing while achieving a faster response time than disk queuing alone. In order to achieve main-storage queuing with disk backup, however, the user must define at least two message queues data sets—one residing in main storage, the other on reusable or nonreusable disk.

Specifying Channel Program Blocks

Channel program blocks (CPBs) are used to transfer data between buffers and direct-access secondary-storage devices. A CPB consists of 64 bytes of control information plus a work area the size of one buffer unit. One CPB is involved whenever the contents of a buffer unit are written on disk or read from disk.

CPBs that are not being used currently are queued in a CPB *free pool.* When a CPB is to be used in writing data onto disk, TCAM "swaps" the CPB with a full buffer unit (the contents of which are to be written onto the disk); that is, the CPB work area is assigned to the available unit queue and a full buffer unit is assigned to the CPB to replace the work area. This swapping of units is accomplished by changing addresses internally; no movement of data occurs.

When the CPB has been used in reading from disk, its full work area is swapped with an empty unit; that is, the CPB work area is assigned to the outgoing group of the Message Handler for the destination, and is replaced by a unit from the available unit queue. Thus, the unit pool always has the same number of units, even though they are not necessarily the same units that were originally in the pool. The number of work areas assigned to the CPB is also constant, although some of the work areas were once buffer units. This swapping feature saves time; when swapping occurs, data need not be moved from the CPB unit into the buffer unit.

NOTE: Swapping does not occur for units involved in a data transfer resulting from disparity in size between origin buffers and destination buffers (for a discussion of such data transfer, see *Other Buffer Design Considerations* in the *Defining Buffers* chapter. In this case, data is moved from the CPB unit to an empty unit.

The number of CPBs in a TCAM system is specified by the CPB= operand of the INTRO macro. The number that should be specified must be determined experimentally, and depends upon the amount of message traffic during peak period of activity in the TCAM system. The following formula may be used to determine initially how many CPBs to specify in a system:

$$\frac{(2(BU) + 1)m}{60} + r$$

where r is 1 if reusable disk and 0 otherwise, m is the average number of messages being transmitted per minute during peak periods of message transmission, B is the number of buffers per message, and U is the number of units per buffer. The maximum number of CPBs that TCAM can use at any one time can be determined by adding the number of units per buffer for every destination QCB in the system (destination QCBs are generated when TERMINAL and TPROCESS macro instructions define stations and application programs to which messages may be directed). There is not much likelihood that TCAM will need this maximum number of CPBs.

If any messages are to be placed in a queue in a data set residing on reusable disk, at least two CPBs must be specified (if any form of disk queuing other than reusable is being used, the minimum CPBs that may be specified is one). It is highly recommended that at least as many CPBs be specified as the maximum number of buffer units per buffer in the system, so that an entire buffer can be dispatched with a minimum number of operations.

Specification of too few CPBs results in poor disk performance; messages are delayed while TCAM waits for CPBs to become available to place the messages upon or remove them from disk. Specification of too many CPBs results in waste of main storage; each CPB has a length of 64 bytes plus the length of a buffer unit.

How to Determine if Too Many CPBs Were Specified on the CPB= Operand of the INTRO Macro Instruction: The type of queuing used by the CPB free pool is LIFO (last-in-first-out), so that any unused CPBs at the bottom of the queue remain in the state they were in at TCAM initialization time (all zeros).

The IEDFCPB field of the AVT points to the first entry in the CPB free pool; the eleventh word of each CPB points to the next lower CPB entry on the queue. Consequently, a dump can be taken before the MCP is closed down, and by tracing the CPBs until one is found in the dump whose first few words are zeros, the user can determine if too many CPBs were specified. For instance, if 50 CPBs were specified, and the first several words of CPB number 22 in the chain contained all zeros, then 29 of the 50 CPBs were not used. If the next execution of this same TCAM MCP is likely to be under the same line and traffic conditions, specifying 25 CPBs should be adequate.

How to Determine if Too Few CPBs Were Specified on the CPB= Operand of the INTRO Macro Instruction: If, as a result of tracing CPBs in the dump discussed above, there are found no CPBs whose first few words are zeros, one of two conclusions can be drawn:

1. The exact number of CPBs required to avoid poor disk performance were specified (that is, all the CPBs were being used simultaneously on at least one occasion during the execution of this MCP so that there was no delay in message traffic to or from disk).
2. More likely, not enough CPBs were specified so that on one or more occasions, TCAM had to wait until a CPB was available before it could place a message on (or remove it from) disk.

The user should increase the number of CPBs the next time he executes this TCAM MCP under the same line and traffic conditions. He can then determine, by the technique described in the previous section, whether the increased number of CPBs is sufficient (if the CPB at the bottom of the CPB free pool does not contain all zeros, then specify a larger number of CPBs the next time this MCP executes).

Preformatting DASD Message Queues Data Sets

Before the Message Control Program is started, TCAM expects message queues data sets on both reusable and nonreusable disk to be totally preformatted by the IEDQXA utility described in the chapter *System Preparation.* The records, into which each disk queue is segmented, should have the same length as that specified by the KEYLEN= operand of the INTRO macro. The name of the disk message queues data set is originally specified on the IEDQDATA DD statement for the IEDQXA program. The data set may be cataloged when the IEDQXA job is run.

Message queues data sets located on disk should be preformatted prior to each cold restart of the MCP.

Using Multiple Arm Support

Increased disk efficiency can be obtained by spreading the disk message queues data set over several volumes (up to 16 volumes per disk data set). At initialization time, this is indicated by listing several volumes on the IEDQDATA DD card for the IEDQXA utility. Each volume so indicated is initialized to contain one contiguous extent of the data set, each volume containing identical amounts of record space for the disk message queues.

At TCAM open time, the old disk message queues data set is recognized as existing on several volumes. OPEN builds an Input/Output Block for each extent, permitting TCAM to issue several EXCP instructions, one per Input/Output Block or extent. When the I/O Supervisor has several EXCP instructions to act upon, increased disk performance is realized by overlapping seek times on the various devices; i.e., one drive can be seeking a cylinder while another drive is actively transferring data. Even better performance can be obtained by having the various volumes mounted on drives supported by different channels. This permits simultaneous search/read-write activity on more than one volume. Records are not assigned sequentially from beginning to end of the data set (although it was initially created sequentially). The record assignment algorithm uses the records of the first track, first cylinder, first extent, in a sequential manner. At the end of that track, instead of progressing to the next track of that same cylinder, records are assigned from

the first track, first cylinder, of the *second* volume. Only one track of each volume is used before going to a new track on the next volume. This permits I/O requests to be made from more than just one volume, thus gaining the advantages of multiple EXCPs on several channels.

The algorithm continues assigning the first track to new volumes until all volumes have used one track. Record assignment returns to the first volume, *second* track, first cylinder. Again, a new volume is used each time the end of a track is reached. This cycle repeats until the first cylinder of all volumes is assigned. Then the second cylinder is similarly assigned and so on until the entire data set is filled.

This procedure is used for both reusable and nonreusable disk message queues.

When reusable disk queuing is used, multiple arm support increases the likelihood that one arm will be reading while another is writing, thus improving the efficiency of the system. However, this advantage may be offset by the need to construct an extra IOB and DEB extent for each volume, thereby increasing the amount of main storage required for the TCAM program.

Reusable Disk Queues

A DASD message queues data set that is reusable can often handle the same amount of message traffic as a nonreusable message queues data set while occupying less disk space than the nonreusable data set. A message queues data set located on reusable disk never runs out of disk space under normal conditions, and the TCAM system need never be closed down to replenish disk space for such a data set. In addition, when reusable disk queuing is used, messages for an inoperative terminal need not be trapped in the data set until the terminal is fixed, but may be sent to an alternate destination (specified by the ALTDEST= operand of a TERMINAL macro), which might be another terminal in close physical proximity to the first. This capability of automatically sending a message to an alternate destination is available only to the user of reusable disk queuing. A somewhat similar capability is provided by specifying a cascade list as a destination (see the description of the cascade list in *Constructing the Terminal Table* in the chapter *Defining Terminal and Line Control Areas*).

A reusable data set requires periodic reorganization. TCAM's method of reorganizing the reusable data set is illustrated in Figure 7.

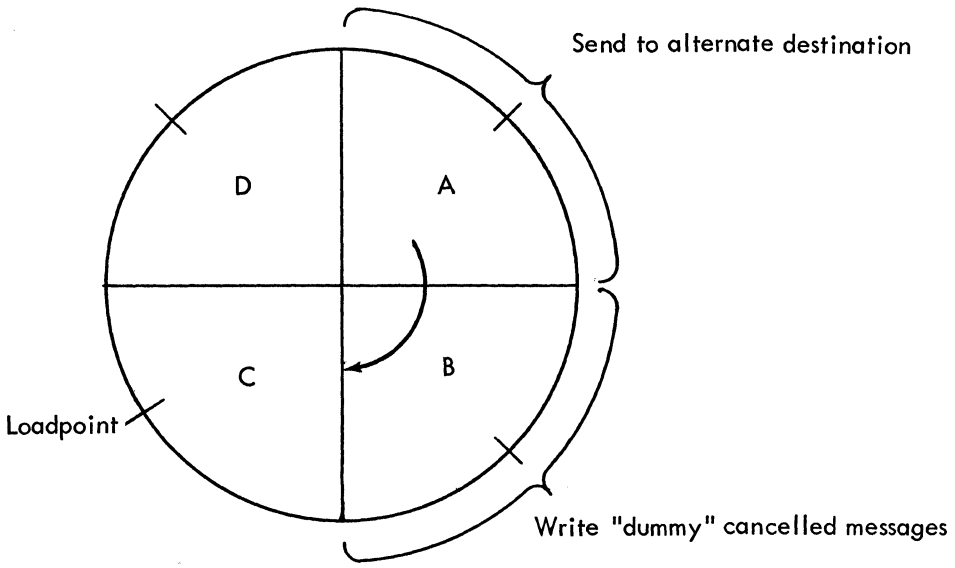


Figure 7. Reorganizing a Reusable Data Set.

The data set as a whole (whether on one volume or sixteen) is divided into four equal zones (shown in Figure 7 as zones A, B, C, and D). Messages are read into the four zones sequentially. By the time Zone D is full, Zone A has been prepared for reuse, and a cycle of use and reuse of the data set has been initiated.

Figure 7 shows a "loadpoint" located half-way through each zone. Assume that the data set has been in use for some time; Zones A and B contain messages received relatively recently. When the loadpoint for Zone C is reached, a TCAM reorganization routine is automatically activated. This routine checks Zone A for messages that have not yet been sent or canceled. Such messages are placed on the queue for the alternate destination specified by the ALTDEST= operand of the TERMINAL or TPROCESS macro for the original destination (these macros and their functions are described in the chapter *Defining Terminal and Line Control Areas*). The alternate destination specified in ALTDEST= may be the original destination. If the alternate destination queue is located in the message queues data set currently being reorganized, the unserviced message is written in Zone C. If any destination QCBs have assigned *next header* records in Zone B, a canceled header is written in this location, thus updating these next header positions to the current zone. This prevents a new message being sent to its alternate destination because its header is too far back (by definition, this is an old message). By the time that the end of Zone D is reached, Zone A is ready for reuse; all unserviced, uncanceled messages that were in Zone A have been copied into Zone C (if the queue for the alternate destination is located in this data set) or copied into another data set. When Zone A is reached, its contents are overlaid with incoming messages. The cycle is repeated as each of the four loadpoints is reached.

NOTE: When a zone is reorganized and the unserviced messages for a particular destination located in that zone are requeued for their alternate destination, they are assigned a message priority equal to or less than their original priority as specified for the alternate destination and are placed in its FEFO queue. For instance, if the original destination message had a priority of 8, and the available priority levels for the alternate destination are 9, 7, 5 and 0, the message will be requeued with a priority level of 7. If the alternate also had a priority level of 8, the original message will be requeued at the same priority level. Messages are sent in the FEFO sending order usually in effect for messages having the same priority on a destination queue (see the discussion of message priority and queuing in the chapter *Defining Terminal and Line Control Areas*). Whether or not this modified message priority and sending scheme for requeued messages turns out to be an asset or a liability to the reusable disk user depends upon his application.

NOTE: When messages are moved in a zone reorganization, sequential ordering is not maintained. The log message queue is not moved. If a backlog develops, logged messages are overlaid and lost.

The advantages of reusable disk queuing have already been mentioned. When the reorganization scheme just outlined is considered, certain disadvantages become evident:

1. The disk activity required during data-set reorganization may result in longer response times than would occur if nonreusable disk were used. Each message that is requeued must be read into main storage and rewritten in a message queues data set, and each dummy canceled message must be written from main storage into the data set on reusable disk. The longer messages remain enqueued on the disk before being sent, the more likely it becomes that they will have to be reread and rewritten. Messages are more likely to linger in a reusable disk queue when the transmission priority for the nonswitched line to their destination is equal or receive rather than send (see the discussion of transmission priority in the chapter *Defining Terminal and Line Control Areas*), when many stations are assigned to the same line, when traffic to a destination is heavy rather than light, when few CPBs are specified, when the destination station is a start-stop rather than a bisynchronous terminal, when the destination is an application program whose data sets are not open, and when a destination station on an Inward WATS line calls the computer relatively infrequently. Terminal reliability is also a factor; if messages for a station must be intercepted by an operator command or by a HOLD macro because the station is inoperative, response time will lengthen as the number of intercepted messages increases; this effect is compounded if the queue for the alternate destination specified for an intercepted station is also located in the reusable disk data set.

2. TCAM's capability of retrieving messages that have already been sent (as described in the *Message Retrieval* section of the chapter *Writing TCAM-Compatible Application Programs*) is limited when reusable disk queuing is used, because the original copy of a transmitted message is eventually overlaid by another incoming message.

A serious problem may arise if a data set on a reusable disk becomes full, i.e., if TCAM's reusability routine is called to service a new zone but has not yet completed servicing the previous zone. If message traffic for the reusability routine to copy to alternate destinations is so heavy that active disk records may be overlaid, a logical read error occurs and TCAM terminates abnormally with a system ABEND code of 045 and a user code of 02 or 03.

This heavy usage of reusable disk may be the result of either a sudden surge of incoming traffic for this queue type, or accumulation of a large number of messages that must be routed to alternate destinations because their primary destinations are unable to accept them.

In an attempt to prevent the need for abnormal closedown, TCAM requests cessation of incoming traffic, permitting send operations to have temporary priority and to clear the data set of unsent messages. When the overlay danger is past, normal receive operations are resumed. If the temporary halt of receive operations cannot prevent overlay of active records, the ABEND is issued.

To reduce the frequency of this slowdown, the following steps may be taken:

a. Format a larger reusable disk data set. As a rule of thumb, the data set should be at least large enough so that the longest message to that data set will span less than a fourth of the disk (less than one of the four zones). Otherwise, the internal TCAM zone reorganization routine may be unable to transmit unsent messages to their alternate destinations (because a zone for the abnormally long message has been overlaid, resulting in loss of header information needed to send this message to its alternate destination).
b. Spread reusable disk data sets over several volumes (and ideally over several channels), thereby facilitating more rapid servicing of the zones by TCAM's reusability routine.
c. If it is likely that a station will be intercepted or otherwise unable to receive an appreciable percentage of the time, do not locate the destination queue for that station on reusable disk.
d. To avoid trapping unsendable messages queued to a defective station, do not specify a station as its own alternate destination.
e. To avoid accumulating messages queued to switched stations, exercise care in the specification of the DIALNO= and CLOCK= operands on the TERMINAL macro. By coding DIALNO=NONE, you prohibit the CPU from initiating a call to send messages to this station. The CLOCK= operand restricts the CPU to a single call every 24 hours.
f. Consider the number of priority levels specified in the TERMINAL macro for each destination queue. Each priority level requires one record for the next header being sent to that destination. Thus, the more priorities that are assigned, the larger the reusable disk needs to be. The number of priorities in the system should be less than one-eighth the total number of records on the disk. To determine the number of priorities, the following formula can be used:

   $T \geq 8(x + y)$

   where T is the total number of records on the disk, x is the total number of TERMINAL, TPROCESS and LOGTYPE macros coded in the terminal table, and y is the number of levels specified in every LEVEL= operand for every TERMINAL and TPROCESS macro defined.
g. Receive priority with too short an interval can cause messages to accumulate and create additional overhead.
h. Finally, remember that the busier the lines, the larger the reusable disk data set should be; turnaround time for message transmission is adversely affected if the data set is not large enough for high-density message traffic.

When using initiate mode or program-controlled interrupt for input, be aware of the possibility that the first segment of a very long message can be overlaid before the last segment is received.

Nonreusable Disk Queues

For a TCAM MCP that must run continuously for an extended period of time with fairly heavy message traffic, the user would have to allocate more disk space if he used nonreusable queuing than he would if he used reusable queuing. In addition, a TCAM system using nonreusable disk queues must be closed down from time to time as the available space in the data set is exhausted. One great advantage that nonreusable disk queues have when compared with reusable disk queues is that system overhead is cut down during extended periods of high message traffic when nonreusable disk queuing is specified, because the data-set reorganization described above for reusable disk queues is not performed for nonreusable disk queues. Nonreusable disk queuing is attractive for applications in which it is likely that many messages will remain enqueued for a relatively long period of time before being sent; general criteria for estimating this likelihood are given above in the discussion of reusable disk queues.

When a certain percentage of the records in a message queues data set on nonreusable disk have been used, a flush closedown (defined in the chapter *Activating and Deactivating the Message Control Program*) is initiated by TCAM. The threshold percentage is specified by the THRESH= operand of the message queues DCB macro (described below) and should be based on a consideration of the maximum number of message units that will result from messages entered at stations and the number of units on the disk data set. If the data set becomes filled before closedown can complete and wraparound of the nonreusable disk will cause the first record to be overlaid, the TCAM MCP will terminate abnormally with a system code of 045 and a user code of 01. Following the flush closedown, the data set must be reformated (using the IEDQXA utility described in the chapter *System Preparation*), and the system may be restarted by means of a cold restart.

Main Storage Queuing

The main-storage message queues data set (if specified) is created at the time the INTRO macro is executed, when an area of main storage is allocated to the buffer-unit pool. The data set resides in the buffer-unit pool, which is described in the chapter *Defining Buffers*. Buffer units containing data directed to a destination queue in the main-storage data set are assigned directly to the appropriate queue. Upon removal from the queue, the units are available for reuse. No data is moved when units are placed on the queue; however, when the message is to be sent to its destination, it is copied from the enqueued units containing it into a buffer. The original copy is held in the queue until the message has been transmitted and any macros in the outmessage subgroup handling it have been given an opportunity to check the message error record for the message; this is done so that the message header may be retrieved from the queue, if necessary.

Because data in main storage is accessed and manipulated more rapidly than data stored on disk, during periods of high message traffic messages directed to destinations whose queues are located in main storage will be received much more rapidly than would be the case if the queues for these destinations were located on disk. Because allocation and deallocation of units for the main-storage data set is dynamic, the data set is essentially "reusable."

When a message queues data set is located in main storage without disk backup, sufficient main storage must be allocated to the data set to handle peak message traffic. The MSUNITS= operand of the INTRO macro specifies the maximum number of units that can be assigned at any one time to the main-storage message queues data set. When insertion of buffer units containing part of an incoming message into the destination queue would cause the number specified by MSUNITS= to be exceeded, bit 8 of all message error records in the system is turned on. The first unit of the buffer is placed in the queue; all other units are lost. If this was not the last buffer in the message, any error handling macros coded in the inmessage subgroup of the Message Handler for this line group that test bit 8 of the message error record are activated. For example, the user might code a MSGGEN or ERRORMSG macro to advise the terminal operator or an application program that message segments are being lost due to a lack of available main-storage units. The operator or application program could then slow down incoming message traffic by means of appropriate operator commands or network control macros until sufficient main-storage units are available. If the segment rejected was the last segment, the entire message (except for the first unit) is lost; in this case the user may test bit 8 of the message error record when another message is handled by this or another Message Handler.

The first unit of a message that is lost due to a lack of main-storage units is always enqueued in its proper destination queue. When this unit is processed by the outgoing group of the Message Handler for the destination station, bit 16 of the message error

record for this message is turned on. In his outmessage subgroup, the user may code error-handling macros to test bit 16. For example, he might code an ERRORMSG macro that would return the unit to the originating station together with a request that this message be retransmitted.

In the event that there is not even enough main-storage space available to permit the first unit of a message to be enqueued, (i.e., if enqueuing the unit would cause the percentage specified in the MSMAX= operand to be exceeded), TCAM nevertheless enqueues the unit. In addition, TCAM refuses to accept any more incoming messages (these messages are not lost) until the number of units in the main-storage data set falls to or below the level specified by the MSMIN= operand of the INTRO macro.

TCAM's only criterion in determining whether units are available for main-storage queuing is the number specified by the MSUNITS= operand of the INTRO macro. It is up to the user to specify a satisfactory number of main-storage units for his system. If he does not, and if no disk backup is provided, throughput will suffer because fewer incoming messages will be accepted, and some message segments may be lost.

TCAM provides the user of main-storage queues with a means of informing himself when the main-storage data set is in danger of running out of units. In the MSMAX= operand of the INTRO macro, the user may specify a percentage of his main-storage data set units (i.e., a percentage of the number specified in the MSUNITS= operand of INTRO); when this percentage of units is enqueued, bit 9 of all message error records in the system is set. The user may code a MSGGEN or ERRORMSG macro in his Message Handlers to check this bit and inform the operator or an application program to slow down invitation until a suitable number of enqueued messages have been sent to their destinations. Since messages for inactive application programs are maintained on main-storage queues, the user may also activate the application program and allow enqueued messages to be sent. The MSMIN= operand of INTRO also specifies a percentage of the total number of units available for main-storage queuing. When the percentage of units enqueued in the main-storage data set falls below that specified by MSMIN=, bit 8 is set on in all message error records in the system. The user may code a MSGGEN or an ERRORMSG macro in his Message Handlers to test this bit and inform the operator or an application program that there is no longer a shortage of main-storage units, so that normal invitation may be resumed.

When the percentage of enqueued units falls below that specified by MSMAX=, bit 9 is turned off in all message error records. When the percentage of enqueued units rises above that specified by MSMIN=, bit 8 is turned off in all message error records.

Neither the intercept function (see the description of the HOLD macro) nor the retrieve capability (see the description of the POINT macro) is possible for messages queued in main-storage-only queues. The ERRORMSG and REDIRECT macros provide a limited retrieval capability when certain errors (such as transmission errors, as detected by the EOB checking facility provided by the STARTMH macro) occur.

A message queues data set located in main storage without disk backup cannot be checkpointed; if the TCAM system closes down or fails, all data in the data set is lost.

Instead of (or in addition to) specifying a main-storage message queues data set with no disk backup, the user may specify a main-storage data set having backup on reusable or nonreusable disk. Main-storage queuing with disk backup combines advantages of disk and main-storage queuing, and avoids certain of the problems associated with the other queuing methods. Data directed to a main-storage queue with disk backup is never lost because of unavailability of main-storage units, and TCAM does not refuse to accept messages when the main-storage data set is full. TCAM's message-interception and message-retrieval functions may be utilized, and closedown and restart of the system without loss of data is possible. Response time is better than with disk queuing, because most outgoing messages do not have to be read from disk.

When main-storage queuing with disk backup is used, TCAM copies each unit arriving at a main-storage queue onto disk. Copying involves a movement of all data in the unit and a writing operation. When the number of units specified in the MSUNITS= operand of the INTRO macro is enqueued in main storage (i.e., when the main-storage queues will accept no more units), data is not lost as it is when main-storage-only queuing is specified; instead, the contents of incoming units are written directly onto disk. No bit in the

message error record is set when main-storage units are exhausted, and invitation is not suspended. The user may utilize the MSMAX= operand of the INTRO macro to warn him that the number of units enqueued in main storage is approaching the maximum permitted, and he may use the MSMIN= operand of INTRO to inform him when the number of units enqueued in main storage has fallen to a safe level.

Outgoing messages are sent from the main-storage queue when they are on this queue; otherwise they are brought in from disk and sent. When a message is sent out from main storage, its copy on disk is marked serviced.

The TCAM intercept function (using the HOLD macro) and retrieve function (using the POINT macro) may be implemented when main-storage queuing with disk backup is used. The disk queues are accessed to provide these functions.

Main-storage queuing with backup on disk uses more main storage than disk queuing and results in a longer response time than would be the case if main-storage queuing with no disk backup were specified (because each message must be completely copied onto disk before it can be sent to its destination). Yet this method of queuing combines many of the attractive features of the other methods, and for many applications it provides an acceptable compromise between the speed of main-storage-only queuing and the reliability of disk queuing.

Specifying One or More Queuing Methods

The user may specify up to three message queues data sets for his TCAM system. One of these resides in main storage, another on reusable disk, while a third is located on nonreusable disk. Taken singly or in combination, these three possible data sets provide the five queuing methods discussed. For main-storage-only queuing, a main-storage data set is needed. Reusable and nonreusable disk queuing each require a data set. If the user wishes to implement main storage queuing with reusable disk backup, he must define two data sets—one in main storage and the other on reusable disk. Two data sets are also required to support main-storage queuing with backup on nonreusable disk.

A TCAM system having two message queues data sets, one in main-storage and one on reusable disk, will support three types of queuing: main-storage-only queuing, reusable disk queuing, and main-storage queuing with backup on reusable disk. The type of queuing used for a particular message in his system depends upon the message's destination. The QUEUES= operand of the TERMINAL or TPROCESS macro defining a remote station or an application program specifies the type of queuing for messages destined for that station or application program. In the system being considered, messages sent to a terminal whose TERMINAL macro specified QUEUES=MO would be queued in main storage only. Messages sent to a terminal whose TERMINAL macro specified QUEUES=DR would be queued on the reusable disk only. Messages sent to a terminal whose TERMINAL macro specified QUEUES=MR would be placed on a queue in main storage if possible, and would also be placed on a queue in the reusable disk data set. Such messages would be retrieved and sent from the main-storage queue, if possible, or from the disk queue.

The number of data sets that must be defined depends upon the type of queuing desired, which in turn depends upon the application. As an example, consider a savings bank inquiry application in which relatively short incoming messages (consisting perhaps of an account number, a transaction amount, and a transaction code) are sent to an application program, and relatively long response messages are returned by the application program. The TCAM user with such an application might wish to use main-storage-only queuing for the inquiry messages (since they are short), and disk queuing for the response messages (since they are long). In this way, he could take advantage of the speed of main-storage-only queuing for his short input messages without giving up the main storage required if main storage queuing were used for his long response messages. To implement his queuing scheme, this user would have to define two data sets—one in main storage, and one on disk.

Checklists for specifying the three types of message queues data sets supported by TCAM follow. The macros listed are discussed elsewhere in this publication. Note that no DD statement or DCB macro is required in defining a main-storage message queues data set, but that both a DD statement and a DCB macro must be issued in defining a message queues data set on either reusable or nonreusable disk. The DCB macro for a message queues data set on disk and the DD statement for such a data set are described in the next two sections.

Checklist for Main-Storage Data Set

Macro	*Operand*	*Comments*
INTRO	MSUNITS=integer	Specifies the maximum number of main-storage buffer units that may be used for queuing.
INTRO	MSMAX=integer	*integer* is a percentage of the number of units specified in the MSUNITS= operand; when this percentage of units is enqueued in the main-storage message queues data set, a bit is set in each message error record in the system to warn the user that his data set is nearly full.
INTRO	MSMIN=integer	When the percentage of the number of units specified by the MSUNITS= operand falls below that specified by MSMIN=, a bit is set in every message error record in the system. This operand may be used to inform the user that his main-storage message queues data set is no longer crowded.
INTRO	DISK={NO / YES}	Specify NO if no message queues data set is to be provided on disk. Specify YES if the system will have a message queues data set on disk.
TERMINAL or TPROCESS	QUEUES={MO / MR / MN}	Code MO for each station or application program whose queues are to be located in main storage only; code MR if backup is to be provided on reusable disk; code MN if backup is to be provided on nonreusable disk.

Checklist for Reusable Disk Data Set

Macro or DD Card	*Operand*	*Comments*
DD Statement		One needed; see *DD Statements for Message Queues Data Sets.*
Message Queues DCB		One needed; see next section.
INTRO	DISK=YES	
INTRO	CPB=integer	Specifies the nonzero number of channel program blocks to be provided for transfer of data between buffers and the disk; see *Specifying Channel Program Blocks* in this chapter.
TERMINAL or TPROCESS	QUEUES={DR / MR}	Code DR for each station or application program whose queues are maintained on reusable disk only; code MR for each station or program whose queues are maintained in main storage with reusable disk backup.

Checklist for Nonreusable Disk Data Set

Macro or DD Card	*Operand*	*Comments*
DD Statement		One needed; see *DD Statements for Message Queues Data Sets.*
Message Queues DCB		One needed; see next section.
INTRO	DISK=YES	
INTRO	CPB=integer	Specifies the nonzero number of channel program blocks to be provided for transfer of data between buffers and the disk; see *Specifying Channel Program Blocks* in this chapter.
TERMINAL or TPROCESS	QUEUES={DN / MN}	Code DN for each station or application program whose queues are maintained on nonreusable disk only; code MN for each station whose queues are maintained in main storage with nonreusable disk backup.

Message Queues DCB Macro Instruction

The message queues DCB macro:

- Defines a message queues data set residing on reusable or nonreusable disk,
- Specifies the location of the data set,
- Specifies the percentage of records in a nonreusable disk data set to be filled before a flush closedown is initiated,
- Is not issued for a message queues data set residing in main storage.

The message queues DCB macro defines a data control block for a message queues data set. Parameters based on the keyword operands specified in the macro are included in the data control block. One message queues DCB macro is required for a message queues data set residing on reusable disk, and one is required for such a data set residing on non-reusable disk. The macro generates no executable code.

The message queues DCB macro has the following format:

Name	*Operation*	*Operands*
diskdcb	DCB	keyword operands

diskdcb

Function: Specifies the name of the macro instruction and the name of the data control block generated by the expansion of the macro.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

keyword operands

Function: Specifies the operands that can be used.
Format: The operands may be specified in any order and are separated by commas with no intervening blanks.
Notes: The operands for the message queues DCB macro instruction are described below. See *Appendix A* for a description of the format and symbols that define macro operands.

When a parameter can be provided by an alternate source, a symbol appears in the alternate source entry for the operand. When there is no alternate source (i.e., the parameter must be specified by the operand), the alternate source entry states "None." The symbols have the following meanings:

Symbol	*Explanation*
DD	The value of the operand can be provided at execution time by the data definition (DD) card for the data set.
PP	The value of the operand can be provided by the user's problem program any time before the data control block exit at open time.
OE	The value of the operand can be provided by the problem program any time up to and including the data control block exit at open time.
NOTE 1: If DD is specified, OE or PP may also be used. If OE is specified, PP may also be used. For information on providing parameters via DD, see *DD Statement for Message Queues Data Sets.* For information on providing parameters via DD and OE, see *Modifying the Data Control Block* in the OS publication *Data Management Services.* The section *Processing Program Description*, in the same publication, describes the data control block exit that can be taken when OE is specified.	
NOTE 2: The formats of macro illustrations, the symbols used in them, and rules for the interpretation of operand descriptions are all provided in *Appendix A.*	

DSORG=TQ

Alternate Source: None.
Function: Specifies that the data set organization is for the message queues or checkpoint data set.
Default: None. This operand is required.
Format: DSORG=TQ

MACRF=(G,P)

Alternate Source: None.
Function: Specifies that access to that data set is gained with GET and PUT macro instructions.
Default: None. This operand is required.
Format: MACRF=(G,P)

DDNAME=ddname

Alternate Source: PP.
Function: Specifies the name that appears in the DD statement associated with the data control block.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.

OPTCD={L / R}

Alternate Source: DD, PP, OE.
Function: Specifies the location of the data set.
Default: None. This operand is required.
Format: L or R.
Notes: L specifies that the data set is to be on nonreusable disk. R specifies that the data set is to be on reusable disk.

EXLST=name of list

Alternate Source: PP.
Function: Specifies the address of the problem program exit list.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: This operand is required if user label, data control block, user ABEND or block count exits are required. The list must start on a fullword boundary. Its format and contents are shown in the OS *Data Management Services* publication. The user ABEND exit is discussed in the last section of this chapter.

THRESH=integer

Alternate Source: DD, OE, PP.
Function: Specifies the percentage of the nonreusable disk message queue records to be used before a flush closedown of the system is initiated.
Default: For reusable disk queues, specification optional. For nonreusable disk queues, 95.
Format: Unframed decimal integer.
Maximum: 100.
Notes: This operand is meaningful for nonreusable disk queues only.

DD Statements for Message Queues Data Sets

One DD statement is needed for each disk message queues data set. The format of this DD statement is as follows:

```
//ddname DD DSNAME=anyname,DISP=OLD
```

where *ddname* is the name specified by the DDNAME= operand of the DCB macro for this data set, and *anyname* is the name of the data set as specified by the DSNAME= operand of the IEDQDATA DD card for the IEDQXA utility used to preformat disk message queues. If the data set is not cataloged, the UNIT= and VOLUME= operands must be included in the DD statement for the disk message queues data set.

The OPTCD= and THRESH= operands of the message queues DCB macro may be omitted from the DCB macro and specified at execution time by coding the DCB parameter of the DD statement for the message queues data set; i.e., DCB=(OPTCD={L / R}) or DCB=(THRESH=n). Both operands may be specified by coding DCB=(OPTCD={L / R}, THRESH=n). These operands are explained in the preceding section.

No DD statement is issued to define a message queues data set in main storage.

Checkpoint Data Set

The TCAM checkpoint facility provides for records to be taken of the MCP environment from which restart can be made in case of closedown or system failure. This facility is described in the section *Using TCAM Service Facilities.*

The checkpoint data set consists of checkpoint records that are maintained and stored on a direct-access storage device. A DCB macro instruction must be issued to define the data control block for the checkpoint data set if the checkpoint facility is to be used. The DD statement associated with the new checkpoint data set must allot space for these records on the direct-access device used. The direct-access device may be either an IBM 2311 Disk Storage Drive or an IBM 2314 Direct Access Storage Facility.

Checkpoint DCB Macro Instruction

The checkpoint DCB macro

- Defines a checkpoint data set residing on a direct-access storage device.

The checkpoint DCB macro has the following format:

Name	*Operation*	*Operands*
chkptdcb	DCB	keyword operands

chkptdcb

Function: Specifies the name of the macro instruction and the name of the data control block generated by the expansion of the macro.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

keyword operands

Function: Specifies the operands that can be used.
Format: The operands may be specified in any order and are separated by commas with no intervening blanks.
Notes: The operands for the DCB for the data set are described below.

See *Appendix A* for a description of the format and symbols that define macro operands.

When a parameter can be provided by an alternate source, a symbol appears in the alternate source entry for the operand. When there is no alternate source (i.e., the parameter must be specified by the operand), the alternate source entry specifies "None." The symbols have the following meanings:

Symbol	*Explanation*
DD	The value of the operand can be omitted from the DCB macro and provided at execution time by the data definition (DD) card for the data set.
PP	The value of the operand can be provided by the user's problem program any time before the data control block exit at open time.
OE	The value of the operand can be provided by the problem program any time up to and including the data control block exit at open time.
NOTE: The formats of macro illustrations, the symbols used in them, and rules for the interpretation of operand descriptions are all provided in *Appendix A.*	

DSORG=TQ

Function: Specifies that the data set organization is for the message queues or checkpoint data set.
Default: None. This operand is required.
Format: DSORG=TQ

MACRF=(G,P)

Alternate Source: None.
Function: Specifies that access to the data set is gained with GET and PUT macro instructions.
Default: None. This operand is required.
Format: MACRF=(G,P)

DDNAME=ddname

Alternate Source: PP.
Function: Specifies the name that appears in the DD statement associated with the data control block.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.

OPTCD=C

Alternate Source: PP, OE, DD.
Function: Specifies that the data set is for checkpoint records.
Default: None. This operand is required.
Format: OPTCD=C

EXLST=address

Alternate Source: PP.
Function: Specifies the address of the problem program exit list.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: This list must be provided if user label, data control block, or user ABEND exits are required. The list must start on a fullword boundary. Its format and contents are shown in *Data Management Services* publication. The user ABEND exit is discussed in the last section of this chapter.

DD Statement for the Checkpoint Data Set

One DD statement must be issued for the checkpoint data set. If DISP=NEW is coded, this statement must allocate space on DASD for all records in the checkpoint data set. A formula for allocating sufficient space is given in Figure 34. The DISP= parameter must be coded DISP=OLD for a warm restart and may be coded either DISP=OLD or DISP=NEW for a cold restart. If DISP=NEW is coded, STARTUP=CY (or STARTUP=CIY, if I is coded in the STARTUP= operand of INTRO) is assumed for the INTRO macro instruction, regardless of what is coded for the STARTUP= operand at assembly time or at WTOR response time. (The disposition term of DISP= may also be coded.) Any incremental quantity request on the SPACE= parameter is ignored.

The OPTCD= operand of the checkpoint DCB macro may be specified at execution time by coding the DCB parameter of the DD statement for the checkpoint data set as DCB=(OPTCD=C).

A typical DD card used for initial allocation is:

```
//TPCHKPNT DD   DSNAME=CPDS,UNIT=2311,      *
//             VOLUME=SER=111111,           *
//             SPACE=(TRK,(3)),             *
//             DISP=(NEW,CATLG)
```

NOTE: The step containing the DISP=(NEW,CATLG) operand must completely terminate normally for the deallocation routine to perform the catalog function. If such a step is halted by SYSTEM RESET or Master Check, the catalog is not updated. The next use of the data set, with DISP=OLD coded, must either supply the UNIT= and VOL=SER= operands, or the name must have been entered into the catalog using the CATLG command of the IEHPROGM system utility.

A typical DD card used for the same (cataloged) checkpoint data set after initial allocation is:

```
//TPCHKPNT DD   DSNAME=CPDS,DISP=OLD
```

The data set does not have to be cataloged. If it is not cataloged, the data set is allocated by specifying DISP=(NEW,KEEP), and subsequent uses of the data set must contain the UNIT= and VOL=SER= keyword operands to provide the information that would otherwise be in the catalog.

NOTE: There is no utility job to format a checkpoint data set. It is formatted by OPEN at each cold restart.

Log Data Sets

A log data set consists of messages or message segments placed on a secondary storage device for accounting purposes. TCAM's support of the logging function is described in the *Using TCAM Service Facilities* chapter, and in the descriptions of the LOG and LOGTYPE macros. The TCAM logging facility is optional.

One log data set should be defined for each secondary storage device on which messages or message segments may be logged. A log data set is defined by a BSAM DCB macro that is issued with the DCBs defining the line group data sets, the message queues data sets, and the checkpoint data set. The BSAM DCB macro is described in the OS *Supervisor and Data Management Macro Instructions* publication.

In the BSAM DCB macro for a log data set, the user should code the following operands:

Operand	*Comments*
DSORG=PS	
MACRF=W	
DDNAME=symbol	
BLKSIZE=keylen	Replace *keylen* with the value specified in the KEYLEN= operand of the INTRO macro.
RECFM=F	
NCP=integer	Replace *integer* with the maximum number of buffer units that may appear in a buffer.

The SYNAD=address operand of the BSAM DCB macro, where *address* is the name of a user-specified, error-analysis routine to be given control when an uncorrectable I/O error is detected, should also be coded. The error routine must conform to the standards set forth in the discussion of this operand in the *Supervisor and Data Management Macro Instructions* publication. Upon return from the error-analysis routine, the log function continues as if no error had been encountered. All indications of the I/O error must be removed from the DCB, so that the MCP will terminate abnormally when the next WRITE is issued. (The format of the BSAM DCB is described in the OS *System Control Blocks* publication.) If this operand is not specified, the MCP terminates abnormally when a permanent I/O error occurs during the logging operation.

The UNIT= parameter of the DD statement associated with each log DCB macro should specify the address of the appropriate secondary storage device.

User ABEND Exits

The DCB macros for the line group data sets and for the message queues data sets permit specification of a user-written routine to be given control if an OPEN macro fails to open the data set for which the DCB macro is coded. The user routine is specified by coding a special entry in the problem-program exit list named in the EXLST= operand of the appropriate DCB macro. (The format and contents of the problem-program exit list are shown in the publication, *Data Management Services.*) The special entry, called the user ABEND entry, consists of a one-byte code of X'0E' followed by the three-byte address of the user routine.

If the OPEN macro for a particular data set fails to execute properly, and if a user ABEND entry is included in the EXLST= operand of the DCB macro for the data set, the user routine is given control. The user routine should save and restore registers. When control is passed to the user routine, the general registers contain the following information:

Register	*Contents*
0	Error code
1	Options available to the user ABEND routine
2-13	Contents prior to execution of the OPEN macro
14	Return address (must *not* be altered by the exit routine)
15	Address of user-routine entry point

The error code, which occupies the right-hand byte of register 0 (the other 3 bytes are set to 0) tells the user the reason why the OPEN failed. Possible error codes and brief explanations are contained in the list that follows.

The error code is also included in a message directed to the system console when the OPEN macro fails to execute properly.

This message, which is sent even when no user ABEND routine is specified, has the following format:

```
IED008I TCAM OPEN ERROR xxx-y IN DCB dcbname descriptor
```

Here, *xxx-y* is the code referred to in the list of error codes. *dcbname* is the name of the DCB macro for the data set that could not be opened properly. *descriptor* is a single word describing the type of error. This message is discussed in the OS publication *Messages and Codes.*

The following instructions may be coded in the user ABEND routine to return control to TCAM:

```
L   13,4(13)
RETURN (14,12),T,RC=(15)
```

If an OPEN macro fails to execute properly and no user ABEND exit is provided, TCAM issues an ABEND macro to terminate the MCP task.

Error codes returned by TCAM OPEN routines

Error Code (Hex Format)	*Meaning*
01	Not enough main storage is available to build a data extent block for a line-group data set. To correct this situation, specify a larger region (MVT) or partition (MFT) on the JOB DD statement for the MCP.
02	Incompatible stations specified in the same line group. To correct, set up your line groups correctly, reassemble, and reexecute.
03	The Device Class field of the first unit control block (UCB) for a station in the line group specifies something other than telecommunications or graphics. To correct, check the addresses specified in the line group DD statements to be sure that you are specifying valid line addresses.
04	An unsupported control unit was specified for this line group. To correct, check the validity of the line addresses specified by the line group DD statements.
05	The adapter-type and model-code bits in a UCB specify something other than the lines specified by the DD statements for this line group. To correct, check the validity of the line addresses specified by the line group DD statements.
06	The device characteristics specified for stations in this line group are not the correct ones for these stations. To correct, check the validity of the line addresses specified by the line group DD statements.
07	Insufficient main storage is available to build a line control block for a line group. To correct, specify a larger region or partition on the JOB DD statement for the MCP.
08	Insufficient main storage is available to build a station control block for a switched line. To correct, specify a larger region or partition on the JOB DD statement.
09	The binary synchronous interface specified in the data control block does not agree with that specified in a unit control block for a line in this line group. To correct, check the type of interface specified in the INVLIST= operand of the DCB macro against the bit settings specified in the UCBs for each line in this line group.

0A	The TERMINAL or TPROCESS macro specifies disk queuing or a DCB macro for a message queues data set on disk is present, but DISK=YES was not specified in the INTRO macro. Therefore, the AVT is not large enough to support disk queuing. To correct this, specify DISK=YES in the INTRO macro, reassemble, and reexecute.
0B	The key length specified in the DCB macro for a message queues data set does not match that specified in the INTRO macro.
0C	When the IEDQXA utility was used to initialize this data set, multiple volumes were specified; these volumes are not all on the same type of device—either all should be on a 2311 drive, or all should be on a 2314 drive. To correct, reinitialize the data set, being certain that all volumes are located on the same type of device. Then reexecute.
0D	The DCBOPTCD field for this DCB specifies something other than reusable or nonreusable queuing. To correct, check the contents of this field.
0E	A GETMAIN macro was issued by TCAM to obtain main storage to build a data extent block for a message queues data set, but insufficient main storage was available to satisfy the request. To correct, specify a larger region or partition on the JOB JCL statement for the MCP.
0F	A GETMAIN macro was issued by TCAM to obtain main storage to build input/output blocks for a message queues data set, but insufficient main storage was available to satisfy the request. To correct, specify a larger region or partition on the JOB JCL statement for the MCP.
10	The named message queues data set was allocated but not formatted correctly; the last record number written on a track is zero. To correct, format the data set by means of the IEDQXA utility, and rerun the job.
11	No valid UCB addresses were found for this line group; all UCB addresses checked were zero. To correct, specify DD statements with valid UNIT= operands.
12	The sum of the header prefix size plus the number of bytes reserved in the first buffer of each message by the RESERVE= operand of the line group DCB macro is equal to or greater than the size of the buffers assigned to the lines in the group for input; thus, there is no room in the buffers for data. To correct, specify a larger buffer size for input on the BUFIN= operand of the line group DCB macro, reassemble, and rerun the job. (If a DD statement source is used to specify this operand, no reassembly is required.)
13	No data set has been specified for disk or main-storage queuing for a specified terminal.
14	There are no lines in the line group (i.e., one or more of the lines in the line group have not been opened due to some other error).
15	A relative line of 0 (zero) has been specified.

Before control is passed to the user ABEND routine, TCAM sets the bits in the right-hand byte of register 1 to indicate to the ABEND routine what courses of action it may take. The code in register 1 indicates possible user options; the TCAM open routines are set up to work properly if any of the courses of action indicated by the code in register 1 are taken. It is recommended that the user ABEND routine restrict its activities to the

options indicated in register 1. Possible user ABEND alternatives and the codes associated with them are shown in the following list.

Code in Register 1	*Permissible User Options*
X'03'	1. You can abnormally terminate the MCP job—either by issuing an ABEND macro in your subroutine, or by placing a return code of X'02' or higher in the right-hand byte of register 15 and returning control to TCAM. 2. You can tell the TCAM open routine to make no further attempt to open this data set, but to pass control to the next instruction in the MCP. This is done by placing a return code of X'00' in the right-hand byte of register 15 and returning control to TCAM. In this case, your MCP will run with restricted capabilities, since it will not be able to use this data set.
X'07'	1. Same as Option 1 for X'03' code. 2. Same as Option 2 for X'03' code. 3. In activating the lines in a line group data set, a TCAM open routine has found a line on which there are stations incompatible with those found on a previous line or lines. (See *Characteristics of a Line Group* in the chapter *Defining the MCP Data Sets* for the common characteristics that stations and lines in the same line group must have.) When such a line is found, TCAM stops activating lines in the line group. By placing a return code of X'01' in the right-hand byte of register 15 and returning control to TCAM, you direct the open routine to open a modified line group data set consisting of only those lines that had been activated at the time that the line having incompatible stations was encountered. In this case, messages directed to stations or lines that were not activated will be enqueued in a message queues data set, but will never be sent to these stations.

If the user specifies a return code of X'01' in register 15 and the option code passed to him in register 1 was X'03', TCAM immediately takes the ABEND exit again; unless the user routine has code providing for this possibility, a loop will result.

For more information on the DCB exit list and how it is specified, see the OS publication *Data Management Services.*

This chapter describes how to start and restart the TCAM Message Control Program, how to initialize and activate the TCAM data sets, and how to close down the TCAM MCP.

Starting and Restarting TCAM

The TCAM Message Control Program is assembled, link-edited, and executed like any other problem program running under an OS System. Sample Job Control Language for assembling, linkage-editing, and executing the MCP is given in the chapter *Putting the MCP Together.*

The TCAM MCP may be started or restarted by placing the Job Control statements for the execute step in the card reader and activating an OS Reader/Interpreter (by a START command issued at the system console) to read the JCL into the system. Another way to start or restart the MCP is to issue a START command that names a cataloged procedure that causes the MCP to be executed. The chapter on putting the MCP together contains sample code and Job Control statements for implementing both types of startup. The various types of restart available to the TCAM user are described in the *TCAM Checkpoint/Restart Facility* section of the chapter *Using TCAM Service Facilities.*

Initialization and Activation

The INTRO, OPEN, and READY macros are issued as a group; together they constitute the data-set initialization and activation section of the Message Control Program. This section must precede the Message Handler sections of the MCP (see the chapter *Putting the Message Control Program Together*). When the INTRO, OPEN, and READY macros have been executed, the TCAM system is ready to handle message traffic.

As the first macro executed in the Message Control Program, INTRO expects to get control from OS Job Management. INTRO establishes standard entry linkage with Job Management, chains save areas, provides addressability, and saves the start parameter list pointer. If it is desired to insert user-written code (which must not contain any TCAM macros) before INTRO, the Message Control Program (i.e., the code beginning with INTRO) should be called as a subroutine of the inserted user code with register 15 containing the address of INTRO, register 14 containing the address to which the MCP returns upon termination of TCAM, register 13 containing the address of a standard 18-word save area, and register 1 containing the start parameter list pointer as originally passed in register 1 from Job Management.

If the user desires to refer to the PARM field of the EXEC Job Control statement, he may either use the register 1 pointer as passed by Job Management (prior to INTRO execution) to find the PARM field, or after INTRO execution, this same value (in register 1) is stored by INTRO in a local constant area, a fullword tagged IEDSPLPT.

The INTRO macro also creates the Address Vector Table (which is the primary control block of the TCAM system) wherein many system variables are defined. When INTRO is executed, it optionally provides for dynamic redefinition of many of these system variables by interpreting the operator's response to a WTOR message. Once the system variables are defined in the AVT, INTRO continues with system initialization, creating buffers and trace tables, and formatting control blocks.

The OPEN macro completes the initialization of the TCAM data sets and activates them for use. The TCAM data sets that must be activated in the MCP by OPEN macros are those for the message queues, the line groups, the message logs, and checkpoint.

Each data set that is used by the MCP can be opened by a separate OPEN macro, or all data sets of the same type (e.g., all line group data sets) can be opened with one OPEN. If used, the message queues data sets must be opened first, and the checkpoint data set must be opened next. Opening a line group data set causes all lines in the line group to be prepared for operation; the lines optionally may be prepared for message transmission at this time, or activation may be deferred until later (the line is opened idle and later started by the STARTLINE operator command).

The READY macro must be the last instruction in the initialization and activation section of the MCP. When READY has executed, the system is ready to handle message traffic. The expansion of this macro causes a branch to the internal routine that supports the

MCP, where procurement of the first message is awaited. When the first message is procured (either from a terminal or an application program), control is transferred to the MH section of the MCP for handling the message.

Once the MH sections are initially entered after the execution of the READY macro, execution of user-specified code in the MCP is restricted to the Message Handlers; that is, the MH sections are continually reentered to handle messages entering and leaving the computer as long as the MCP is active. Accordingly, any user code must either be within or be branched to from a Message Handler. User code cannot branch between Message Handlers. (See the *User Code in a Message Handler* section of the chapter *Designing the Message Handler.*)

For a sample MCP initialization and activation routine, see the last section of this chapter.

In addition to initial startup of the TCAM system, as described above, TCAM provides for three types of restart following system closedown or failure. These are discussed in *Restart* of the chapter *Using TCAM Service Facilities.*

INTRO Macro Instruction

The INTRO macro:

- Creates the Address Vector Table (the primary control block in the TCAM system);
- Performs the bulk of TCAM system initialization;
- Establishes addressability and entry linkages for the Message Control Program;
- Specifies the name of the Message Control Program;
- Specifies the number of channel program blocks to be provided for transferring data between buffer units and queues maintained on disk;
- Specifies the maximum number of command input blocks that may be used at any one time to contain operator commands entered at the system console;
- Identifies the primary operator control terminal;
- Specifies a character string used to identify operator commands;
- Specifies the size of buffer units;
- Specifies the maximum number of units that may be assigned to a main-storage message queues data set;
- Provides the user with a means of determining when his main-storage message queues data set is nearly full, and when this condition of impending fullness has abated;
- Identifies the station or application program to which messages having an invalid destination are to be forwarded;
- Specifies which user registers are to be saved when in-line user code is located in an inheader or outheader subgroup that may handle multiple-buffer headers;
- Specifies the length of the system interval;
- Specifies the interval between environment checkpoints;
- Specifies the number of environment checkpoint records to be retained at any one time;
- Provides sytem optimization by specifying that unnecessary options are to be omitted;
- Specifies the type of restart to be performed following system closedown or failure;
- Specifies a password that must be coded in certain application-program macros that affect operation of the MCP;
- Provides for inclusion of various debugging facilities;
- Specifies whether a special operator awareness message is to be displayed at the primary operator control station whenever a station fails to respond to polling.

TCAM relies upon the INTRO macro to supply information defining and initializing a variety of TCAM functions. The operands of INTRO provide information concerning data-set definition (DISK=, CPB=, MSUNITS=, MSMAX=, MSMIN=), buffer definition (KEYLEN=, LNUNITS=), the operator control facility (CIB=, PRIMARY=, CONTROL=), the Message Handlers (DLQ=, USEREG=), line control (INTVAL=), the TCAM checkpoint/restart facility (CPINTVL=, CPRCDS=, STARTUP=, RESTART=, CKREQS=), system optimization (CPB=, DISK=, FEATURE=, LINETYP=), network configuration (PASSWD=), debugging aids (TRACE=, TREXIT=, DTRACE=, CROSSRF=, TOPMSG=, COMWRTE=), and the on-line test facility (OLTEST=). A section devoted to each of the above topics is located elsewhere in this publication. The user may read over the description of the INTRO macro for general information before he is familiar with the TCAM functions to which the operands refer, but he should not attempt to code INTRO operands dealing with a particular TCAM function until he has read the discussion of that function. In general, the operand descriptions refer the reader to the sections where the functions of the operands are discussed.

TCAM provides the user with the ability to replace, at INTRO execution time, values specified at assembly time by certain operands of the INTRO macro and to provide values for INTRO operands that were omitted at assembly time.

At the time INTRO is executed, it may cause the following WTOR message to appear on the system console:

nn IED002A SPECIFY TCAM PARAMETERS

This WTOR message is issued only if at least one of the following operands is omitted from the INTRO macro: STARTUP=, KEYLEN=, LNUNITS=, and (if DISK=YES is coded in INTRO) CPB=. If these operands are all coded in the INTRO macro, no WTOR message is issued at execution time.

After the TCAM system issues the WTOR message, it waits for a user response to be entered at the system console. The user has two options in responding; he may either enter response keywords (as shown in the "response keyword" line in the list of INTRO operands.), or he may enter INTRO operand names (as shown in the header line in the list of operands) together with appropriate values. Several keywords or operands may be coded in one response. Keywords or operands coded in a response are separated by commas or vertical bars (|). Responses may be entered in upper- or lower-case letters. They will be translated into upper-case automatically. Each response is limited to 41 characters. After a response has been entered, TCAM re-issues the WTOR message, and continues to issue it after each response is entered until the user indicates that he has nothing more to specify; the user does this by coding U at the end of a response. If the user codes U and has not yet specified values for STARTUP=, KEYLEN=, LNUNITS= or (if DISK=YES is specified in INTRO) CPB=, either in the INTRO macro or in a response to the WTOR message, TCAM prompts him with the following message:

nn IED004A REQUIRED PARAMETER MISSING. SPECIFY operand

where *operand* is the name of the missing INTRO operand.

An error in specifying a response keyword or operand (such as an invalid response keyword or operand or an invalid value with a response keyword) causes an error message to be printed at the console; the operator may respecify the response keyword or operand when he receives such a message. An error in one response keyword or operand prevents interpretation of any keywords in the same response to the right of the keyword in error. A response keyword or operand may be coded more than once in the sequence of WTOR responses; the latest value specified applies.

Example:
The following WTOR messages and responses occur at INTRO execution time for a user who has omitted the STARTUP= and LNUNITS= operands from his INTRO macro. The user specifies LNUNITS=, MSMIN=, MSMAX=, CPRCDS=, and CONTROL=, but forgets to specify STARTUP= (a required operand) and is prompted for this operand.

message: 00 IED002A SPECIFY TCAM PARAMETERS

response: r 00,'B=2,MsMIN=80|X=95,E=5'

message: 00 IED002A SPECIFY TCAM PARAMETERS

response: r 00,'CONTROL=OPcONT,U'

message: 00 IED004A REQUIRED PARAMETER MISSING. SPECIFY STARTUP=

response: r 00,'s=c,u'

NOTE: If no response keyword is shown for a particular operand, the value for that operand may not be specified at INTRO execution time.

The INTRO macro has the following format:

Name	*Operation*	*Operands*
[symbol]	INTRO	keyword operands

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

keyword operands

Function: Specifies the operands that can be used.
Format: The operands may be specified in any order according to assembler language conventions.
Notes: The operands for the INTRO macro are described in the following list.

The list of operands for the INTRO macro also shows the one-character response keywords that may be substitutes for the operand names in responses to the WTOR message SPECIFY TCAM PARAMETERS sent to the system master console at INTRO execution time.

PROGID=characters

Response Keyword: None.
Function: Specifies the name of the Message Control Program.
Default: None. Specification optional.
Format: One to 230 unframed characters with no embedded blanks or commas.
Notes: TCAM inserts this name in a DC C'*characters*' field located in the MCP. In a dump, this name appears in the EBCDIC field at the right of each page of the listing and identifies the beginning of executable code for the MCP. If this operand is omitted, no name is assigned the MCP.

DISK={NO / YES}

Response Keyword: None.
Function: Specifies whether any of the message queues data sets defined for this MCP are located on a direct-access secondary storage device.
Default: DISK=YES
Format: YES or NO.
Notes: DISK=YES is coded if any of the message queues data sets are located on disk. DISK=NO is coded if no message queues data sets are located on disk.

For further information, see *Message Queues Data Sets* in the chapter *Defining Data Sets.*

CPB={integer / 0}

Response Keyword: D=
Function: Specifies the number of channel program blocks to be provided for transferring data between buffer units and message queues maintained on disk.
Default: CPB=0
Format: Unframed decimal integer.
Maximum: 65535
Notes: One CPB is involved in transferring the data in one unit to disk, or in filling one unit with data from disk. See *Specification of Channel Program Blocks* in the chapter *Defining Data Sets.*

This operand is ignored if DISK=NO is coded, and may be omitted in this case. If DISK=NO is coded, this operand is ignored by INTRO, and both CPB= and D= are invalid responses at INTRO execution time.

CIB={integer / 2}

Response Keyword: C=
Function: Specifies the maximum number of command input blocks (CIBs) that can be utilized at any one time in the TCAM system.
Default: CIB=2
Format: Unframed decimal integer.
Maximum: 255

Notes: CIBs are buffer-like areas used to contain operator commands entered at the system console. Space for them is allocated dynamically when needed, and the main-storage space assigned to a CIB 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. However, more than one CIB should be specified if the user anticipates attempting to process simultaneously more than one operator command entered from the console. If an attempt is made to enter an operator command from the system console when the number of CIBs specified by the CIB= operand are present already in the system (because that many operator commands from the system console are now being processed), the message being entered is rejected by TCAM.

PRIMARY={**termname** / **SYSCON**}

Response Keyword: P=
Function: Specifies the name of the station or application program to be used as the primary operator control station.
Default: PRIMARY=SYSCON
Format: *termname* or SYSCON. *termname* is the name of a station or application program (defined by a TERMINAL or TPROCESS macro) and may not be a switched line. It must be able to enter and to accept messages.
Notes: SYSCON is the name of the system console. The functions of the primary operator control station are given in *The Operator Control Facility* section of *Using TCAM Service Facilities.*

If *termname* is changed by a CPRIOPCL operator command, execution of a warm or continuation restart causes the value specified in the macro to be overridden by the value specified by the last CPRIOPCL command executed before closedown or failure.

CONTROL={**characters** / **0**}

Response Keyword: L=
Function: Specifies the character string used to identify each operator command as such to TCAM.
Default: CONTROL=0
Format: One to eight unframed characters with no embedded commas or blanks.
Notes: CONTROL=0 indicates that no character string is being specified, and is valid only if all operator commands are to be entered at the system console.

KEYLEN=integer

Response Keyword: K=
Function: Specifies the size of a buffer unit.
Default: None. This operand is required.
Format: An unframed decimal integer greater than 32.
Maximum: 255
Notes: Guidelines for coding this operand are given in the chapter *Defining Buffers.* This chapter should be thoroughly understood before KEYLEN= is specified.

If disk queuing is used, integer must be identical with the unit size specified in the DCB=(KEYLEN=unitsize) parameter of the IEDQDATA DD statement for the IEDQXA utility program used to preformat the disk queues.

A buffer must be large enough to accommodate the larger of:

a. a header prefix (30 bytes) plus the maximum number of reserve characters specified for the first buffer by the RESERVE= operand of any line group DCB macro or PCB macro plus three bytes, or
b. a text prefix (23 bytes) plus the maximum number of reserve bytes specified for buffers other than the first by the RESERVE= operand of any line group DCB macro or PCB macro plus three bytes.

UNITSZ=integer

Response Keyword: K=
Function: Specifies the size of a buffer unit.
Default: None. If KEYLEN= is specified, this operand must be omitted. If KEYLEN= is not specified, this operand is required.
Format: An unframed decimal integer greater than 32.
Maximum: 255
Notes: This operand is an alternate spelling of the KEYLEN= operand. Either form, but not both, may be specified.

LNUNITS=integer

Reponse Keyword: B=
Function: Specifies the number of buffer units that may be used in building buffers to contain incoming and outgoing message segments.
Default: None. This operand is required.
Format: Unframed decimal integer greater than zero.
Maximum: 65535
Notes: Guidelines for coding this operand are given in the chapter *Defining Buffers.*

MSUNITS={integer / 0}

Response Keyword: M=
Function: Specifies the maximum number of buffer units that may be assigned to the main-storage message queues data set at any one time.
Default: MSUNITS=0
Format: Unframed decimal integer.
Maximum: 65535
Notes: Guidelines for coding this operand are given in the discussion of main-storage message queues data sets in the chapter *Defining Data Sets.*

This value is added to that specified for LNUNITS= to determine the total number of units in the buffer unit pool.

If specified as part of the WTOR response at INTRO execution time, M=0 is considered an invalid response. At execution time, a value greater than zero must be specified.

If MSUNITS=0 is specified or assumed, the expression M= may not be entered in the WTOR response at INTRO execution time. Therefore, if a main-storage message queues data set is desired, MSUNITS= must be coded with a nonzero integer, even if the value specified is to be overridden at INTRO execution time by an M= expression.

Either MSUNITS= must be nonzero or DISK=YES must be coded.

MSMAX={integer / 70}

Response Keyword: X=
Function: Specifies the percentage of the number of units specified by the MSUNITS= operand to be enqueued on a main-storage message queues data set before a warning is provided that the data set is nearly full.
Default: MSMAX=70
Format: An unframed decimal integer greater than zero.
Maximum: 100
Notes: When this percentage of units is enqueued, a bit is set in each message error record in the system. This operand is discussed in greater detail in the section on main-storage message queues data sets in the chapter *Defining Data Sets.*

MSMIN={integer / 50}

Response Keyword: Y=
Function: Specifies the percentage of the number of units enqueued on a message queues data set (specified by the MSUNITS= operand) below which a bit is set in every message error record in the system.
Default: MSMIN=50
Format: An unframed decimal integer.
Maximum: 99
Notes: The operand may be used to inform the user that his message queues data set is no longer crowded. The value specified for MSMIN= must be less than that specified for MSMAX; otherwise, the INTRO macro does not execute. Values specified for MSMIN= (or MSMAX=) at INTRO execution time by means of a WTOR response are checked against the current value of MSMAX= (or MSMIN=) if specified, to ensure that this rule is not broken. If the rule is broken the value specified in the WTOR response is rejected and an error message is sent to the system master console informing the operator of this fact. The operator may then respecify the value. As an example, if MSMIN=95 and MSMAX=99 are coded in the INTRO macro, an INTRO execution time the user should not code

```
r 00,'MSMAX=90,MSMIN=85'
```

as a WTOR response, because the WTOR response is read from left to right and the new MSMAX value will be compared with the old MSMIN value and be rejected. If however, the user codes

r 00,'MSMIN=85,MSMAX=90

these values will be accepted since the new MSMIN value is less than the old MSMAX value, and the new MSMAX value is greater than the new MSMIN value, with which it is compared.

DLQ={**entry** / **0**}

Response Keyword: Q=
Function: Specifies the name of the dead-letter queue to which messages with invalid destinations are sent.
Default: DLQ=0
Format: *entry* is the name of a station or application program as defined by a TERMINAL or TPROCESS macro. DLQ=0 specifies that no dead-letter queue is to be used.
Notes: Dead-letter messages are messages having invalid destinations as determined by a FORWARD macro. If a user-specified routine is coded for the EXIT= operand of the FORWARD macro, messages with invalid destinations may have the destination corrected. If both the DLQ= operand of INTRO and the EXIT= operand of FORWARD are omitted, dead-letter messages are overlaid and lost.

USEREG={**integer** / **0**}

Response Keyword: None.
Function: Specifies the number of registers to be saved when inline user code is located in an inheader or outheader subgroup that may handle multiple-buffer headers.
Default: USEREG=0
Format: An unframed decimal integer.
Maximum: 10
Notes: For guidelines on specifying this operand, see the section *User Code in a Message Handler.* USEREG= specifies sequential registers, beginning with register 2. For instance, if USEREG=4 is coded, registers 2, 3, 4 and 5 are saved.

INTVAL={**integer** / **0**}

Response Keyword: I=
Function: Specifies the number of seconds in the system interval.
Default: INTVAL=0
Format: An unframed decimal integer.
Maximum: 65535
Notes: The system interval is described in *Maintaining Orderly Message Flow* in the chapter *Defining Terminal and Line Control Areas.*

Unless a nonzero integer is specified either in the operand or in the response to a WTOR message at INTRO execution time, no system interval is possible for the MCP.

CPINTVL={**integer** / **1800**}

Response Keyword: V=
Function: Specifies the maximum number of seconds between environment checkpoints when the TCAM checkpoint/restart facility is used.
Default: CPINTVL=1800
Format: An unframed decimal integer greater than 29.
Maximum: 65535
Notes: See the section *TCAM Checkpoint/Restart Facility* of the chapter *TCAM Service Facilities* for further information on this operand.

CPRCDS={**integer** / **2**}

Response Keyword: E=
Function: Specifies the number of environment checkpoint records to be retained in the checkpoint data set at any one time.
Default: CPRCDS=2
Format: An unframed decimal integer greater than 1.
Maximum: 75
Notes: The most recent records are the ones retained. For example, if CPRCDS=2 is specified, the most recent two environment checkpoints are kept in the checkpoint data set. When a new environment checkpoint is taken, its record overlays the oldest environment checkpoint record then being held in the data set. If an attempt is made to increase or decrease *integer* during a warm or continuation restart the smaller value prevails.

Guidelines for coding this operand are included in the discussion of the TCAM checkpoint/restart facility in the chapter *Using TCAM Service Facilities.*

STARTUP={C / CY / W / WY}[I]

Response Keyword: S=
Function: Specifies the type of startup to be performed following closedown of the Message Control Program or system failure.
Default: None. This operand is required.
Format: C, CI, CY, CYI, W, WI, WY, or WYI.
Notes: The types of restart are defined in the discussion of the TCAM checkpoint/restart facility in the chapter *Using TCAM Service Facilities.*

The values may be specified in any order. For instance, IC is just as valid and produces the same results as CI.

C specifies that a cold restart is to be performed following a normal quick close or flush close, and that continuation restart (including scanning of the message queues) is to be performed following system failure.

CY specifies that a cold start is to be performed following a quick close, a flush close or a system failure.

W specifies that a warm restart is to be performed following a normal quick close or flush close, and that a continuation restart is to be performed following system failure. The continuation restart will include full scanning of the message queues.

WY specifies that a warm start is to be performed following a quick or flush close, and that a continuation restart is to be performed following system failure. The continuation restart will not include scanning of the message queues.

I specifies that the status of each invitation list is to be included in the checkpoint record. If I is not coded, invitation lists are not checkpointed. The status information recorded is as follows:

1. whether the list is active or inactive,
2. whether the list is autopolled or program polled.

The specification of I prevails from one cold restart to the next. Attempts to change this specification during a warm or continuation restart are ignored.

CKREQS={integer / **0**}

Response Keyword: R=
Function: Specifies the maximum number of destination queues in use at any time for application programs that include a CKREQ macro.
Default: CKREQS=0
Format: An unframed decimal integer.
Maximum: 255
Notes: This operand specifies the number of checkpoint request records to be set up in a checkpoint data set.

If an attempt is made to increase or to decrease *integer* during a warm or continuation restart, the smaller value prevails.

RESTART={integer / **0**}

Response Keyword: N=
Function: Specifies which environment checkpoint record the TCAM restart facility should use in attempting to reconstruct the MCP environment as it existed at the time of closedown or failure.
Default: RESTART=0
Format: An unframed decimal integer.
Maximum: 255
Notes: For more information on the use of this operand, see the section discussing the checkpoint/restart facility in the chapter *Using TCAM Service Facilities.*

If 0 is specified, the latest environment checkpoint record is used, if 1 is specified, the next to the latest record is used, etc.

Although the maximum that may be specified is 255, the value entered must be less than the number of environment checkpoint records kept, as specified by the CPRCDS= operand. A scan is performed at restart if scanning is specified in the STARTUP= operand. If RESTART=0 is specified, or the operand is omitted and the latest environment checkpoint record cannot be used (due, perhaps, to a disk I/O error), TCAM automatically goes back to the latest usable record and uses it.

If the message queues data set is on resuable disk and the integer specified causes TCAM to attempt to restructure the environment from a checkpoint record that was taken before serviced messages in certain queues were overlaid, it is unlikely that a warm restart or a continuation restart will be successful.

This value should not be changed during a warm or continuation restart.

PASSWRD= {**characters** / **0**}

Response Keyword: W=
Function: Specifies a character string that must be entered in an MRELEASE, MCPCLOSE, TCHNG or ICHNG macro issued in an application program.
Default: PASSWRD=0
Format: One to eight unframed characters with no embedded blanks or commas.
Notes: If this operand is coded, none of the above macros are executed unless they have the correct password. A macro with an incorrect password or no password is ignored.

PASSWRD=0 indicates that no password is being specified. The user will be unable to find the password as specified here in a storage dump; an internal TCAM routine scrambles the password at INTRO execution time.

CROSSRF= {**integer** / **0**}

Response Keyword: F=
Function: Specifies the number of entries in the cross-reference table.
Default: CROSSRF=0
Format: An unframed decimal integer.
Maximum: 65535
Notes: The cross-reference table is a debugging aid. Each entry contains the addresses of certain internal TCAM control blocks associated with a line. If a cross-reference table is to be used, CROSSRF= should specify the maximum number of TCAM lines that are open simultaneously.

This facility is described in the *Debugging Aids* section of the chapter *Using TCAM Service Facilities.*

TRACE= {**integer** / **0**}

Response Keyword: T=
Function: Specifies the number of entries in the TCAM input/output trace table.
Default: TRACE=0
Format: An unframed decimal integer.
Maximum: 65535
Notes: This table provides a sequential record of the I/O interrupts occurring on a specified line and is described in greater detail in *Debugging Aids* in the *Using TCAM Service Facilities* chapter.

TREXIT=symbol

Response Keyword: None.
Function: Specifies the entry point of a user-written routine to be given control when all entries in the TCAM I/O trace table have been used.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: The routine is passed the address of the I/O trace table in register 0. Nothing is returned by the routine. There is no special restriction on what may be done with the table in the routine (i.e., the information might be transferred to an external device such as the printer).

This operand cannot be specified if TRACE=0.

The entries are reusable and may be updated while the exit routine is processing them, since they are updated by code that is disabled to interrupts.

DTRACE={**integer** / **0**}

Response Keyword: A=
Function: Specifies the number of entries in the TCAM dispatcher trace table.
Default: DTRACE=0
Format: An unframed decimal integer.
Maximum: 65535
Notes: The dispatcher trace table is a debugging aid that keeps a sequential record of subtasks activated by the TCAM dispatcher. This table is discussed in *Using TCAM Service Facilities.* One entry is created for each subtask activated; when the end of the table is reached, the table is wrapped and new entries overlay the oldest entries.

OLTEST={**integer** / **10**}

Response Keyword: 0=
Function: Specifies the number of 1024-byte blocks of main storage to be used for on-line test procedures.
Default: OLTEST=10
Format: The unframed decimal integer 0, or an unframed decimal integer greater than 9.
Maximum: 255
Notes: For information on coding this operand, see *On-Line Test Function* in the chapter *Using TCAM Service Facilities.*

If the on-line test capability is not needed, OLTEST=0 should be coded. If the operand is omitted, 10K of main storage is reserved for OLT.

COMWRTE={**YES** / **NO**}

Response Keyword: G=
Function: Specifies that a special task is to be attached to the MCP.
Default: COMWRTE=NO
Format: YES or NO.
Notes: The task must be provided by the IBM Field Engineering representative. If COMWRTE=YES is coded but the task is not available, the MCP runs, but a task associated with COMWRTE terminates abnormally and the COMWRTE function is not available. This operand should be omitted unless the user is directed otherwise by the IBM Field Engineering representative.

WTTONE={**integer** / **0**}

Response Keyword: None.
Function: Specifies the duration of the "mark" character for the World Trade users whose BSC lines require a "line tone."
Default: WTTONE=0
Format: An unframed decimal integer.
Maximum: 450
Notes: Restricted to use in World Trade countries. This operand specifies the number of characters that constitute the tone.

TOPMSG={**NO** / **YES**}

Response Keyword: H=
Function: Specifies whether the operator awareness message IEA001 is to be displayed at the primary operator control station when a polled station fails to respond to polling.
Default: TOPMSG=YES
Notes: The message is described in the section *TCAM I/O Error-Recording Facilities* of the chapter *Using TCAM Service Facilities.*

If YES is specified, the message is displayed each time a station fails to respond to polling during a pass through the invitation list (because, for instance, the station is inoperative).

LINETYP={**BISC** | **STSP** | **MINI** | **BOTH**}

Response Keyword: None.
Function: Specifies the type of lines used in the TCAM system.
Default: LINETYP=BOTH
Format: BISC, STSP, MINI or BOTH.
Notes: BISC is specified if all lines in the system are BSC lines only. STSP is specified if all lines are start-stop only. MINI is specified if all lines in the system are IBM 1050 terminals on leased lines. BOTH is specified if all types of lines are supported.

FEATURE=
({**NODIAL** | **DIAL**},{**NO2741** | **2741**},{**NOTIMER** | **TIMER**})

Response Keyword: None.
Function: Specifies additional features to be supported in the TCAM system.
Default: FEATURE=(DIAL,2741,TIMER)
Format: NODIAL or DIAL, NO2741 or 2741, and NOTIMER or TIMER. Framing parentheses must be coded. If a suboperand other than the last is omitted due to default, a comma must be coded to indicate that it is missing.
Notes: DIAL is specified if dial lines are used. If 2741 terminals are supported, 2741 should be coded. The TIMER suboperand should be specified if any of the following features are included in the system:

checkpoint
any interval
dial-out options
main-storage queuing
reusable disk queuing.

If NOTIMER is specified but a function requiring the timer is used, TCAM terminates abnormally with a system abend code of 045 and a user code of 06.

NOTES: Following the INTRO macro the user should include a section of code that tests the return code in register 15 to determine whether INTRO has executed correctly. If register 15 contains anything other than zero after execution of INTRO, it is unlikely that the MCP will work satisfactorily. See the sample activation and deactivation section of the MCP at the end of this chapter for a section of user code that checks the INTRO return code and branches to an ABEND macro if the return code is anything other than zero.

OPEN Macro Instruction

The OPEN macro:

- Completes initialization and activation of data sets belonging to the Message Control Program,
- Is required for each MCP data set represented by a DCB macro, and for log data sets (if present),
- Specifies whether activation of lines represented by line group data sets is to be immediate or deferred.

OPEN is used to complete initialization and activation of MCP data sets, and to provide an interface with the BSAM routines handling the logging function for TCAM. Each MCP data set required for execution (with the exception of a message queues data set residing in main storage) must be activated in the Message Control Program by an OPEN macro. Log data sets, if present, are also activated by an OPEN macro issued in the MCP. Each MCP data set may be activated by a separate OPEN macro, or all data sets of the same type (e.g., all line group data sets, or all message queues data sets) may be activated as a group by a single OPEN. If message queues data sets residing on disk are present, they must be opened first. The checkpoint data set, if present, must be opened next.

Instead of a standard OPEN macro, the user may code a list and an execute form of the macro, which would be used in conjunction with each other; for general information on the list and the execute form of a macro, including a discussion of the advantages of using these forms, see the OS publication *Supervisor Services.*

When an OPEN macro tries and fails to properly open a TCAM data set, an error message is sent to the system console. This error message, which specifies the data set that could not be satisfactorily opened and also tells why it could not be opened, is described in the section *User ABEND Exits* of the chapter *Defining the MCP Data Sets.* In addition to sending the error message, TCAM provides the user with the capability of specifying a user-written subroutine that receives control when an OPEN macro fails to execute properly. This capability is described in the *User ABEND Exits* section. If the user fails to provide this subroutine, TCAM issues an ABEND macro for the MCP program when an OPEN fails to execute properly.

When an OPEN macro is executed for a line group data set, TCAM issues commands to prepare each line for message traffic. If TCAM does not receive an indication that the commands have successfully executed within 28 seconds from the time they were issued, the line is considered to be temporarily unavailable, and the following message is written at the system console:

IED0791 ENDING STATUS NOT RECEIVED FROM LINE nnn -- LINE UNAVAILABLE

The unavailable line may subsequently be started by the STARTLINE operator command. Unavailability of one line does not affect preparation for message traffic on other lines in the line group.

The operand field of the OPEN macro consists of one or more groups of positional operands, followed by a single keyword operand. Each group of positional operands consists of the name of the data control block for the data set being opened (the name of the block is the name of the DCB macro that created it) and some optional information about the data set being opened. A comma is coded between groups. The optional keyword operand at the end permits the list and the execute form of the macro to be specified.

The OPEN macro has the following format:

Name	*Operation*	*Operands*
[symbol]	OPEN	(dcbname,[({OUTPUT / INOUT / INPUT}) [,IDLE])],...) [,{MF=L / MF=(E,listname)}]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).
Notes: If MF=L is specified, this name must also be provided. It becomes the name of the parameter list generated by the macro.

dcbname

Function: Specifies the name of a data control block identical with the name specified in the *symbol* field of the DCB macro for the data set being opened.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: Register notation may also be used, in which case the specified register (2 through 12) should contain the address of the data control block for the data set being opened.

{OUTPUT / INOUT / INPUT}

Function: Specifies the type of data set with respect to the direction in which message traffic may flow.
Default: INPUT
Format: OUTPUT, INOUT or INPUT.
Notes: OUTPUT specifies an output data set, and must be specified for a log data set. The operand may be coded for a line group data set if none of the lines are to stations that can enter data; in this case, the INVLIST= operand of the line group DCB macro must refer to an invitation list having no entries (see the description of the INVLIST macro).

INOUT specifies a data set that can be used for both input and output. INOUT must be specified for a DASD message queues data set or a checkpoint data set, and should be specified for a line group data set if any of the lines are to stations that both enter and accept data.

INPUT specifies an input data set. This operand may be specified if none of the lines are to stations that can accept data.

IDLE

Function: Specifies whether the lines are to be activated when OPEN is executed.
Default: None. Specification optional.
Format: IDLE
Notes: Is meaningful only for a line group data set. If IDLE is coded, the line group data set is initialized at OPEN execution time, but the lines are not activated. That is, no invitation or selection is performed for stations on this line. Such lines may be activated individually by means of a STARTLINE operator command. If IDLE is omitted, all lines in the line group are automatically activated when the OPEN macro is executed. For nonswitched lines to stations having polling characters, polling of stations having active entries in the invitation lists for the lines begins after OPEN is executed, provided that INPUT or INOUT is also specified.

A station assigned to a switched line that is idle may not call in on that line, but may call in on any active line in its line group and enter messages. Such a station will not receive any messages queued for it until it is activated by the STARTLINE operator command.

If neither INOUT, INPUT, OUTPUT nor IDLE is specified for a particular data set, and a subsequent data control block address is specified in the sublist, two commas must appear between the two specified data control block names.

MF={L / (E,listname)}

Function: Specifies whether the OPEN macro is to generate a parameter list only or is to generate executable code.
Default: None. Specification optional.
Format: *listname* specifies the name of an OPEN macro specifying MF=L.
Notes: MF=L causes creation of a parameter list based on the OPEN operands. No executable code is generated. The user must specify this form of the OPEN among his program constants. The parameters in the list are not used until the program issues an OPEN or CLOSE macro with an MF= (E,listname) operand that refers to the list. The name specified in the name field of the OPEN macro becomes the name assigned to the parameter list.

MF= (E,listname) causes execution of the OPEN routine, using the macro having the MF=L operand specified. Parameters specified through a macro having MF= (E,listname) operand override corresponding parameters in the list.

Example:
The following OPEN macros open:

1. Two DASD message queues data sets whose name (as assigned by their DCB macros) are DISKREUS and DISKNON;
2. A checkpoint data set named TPCHK;
3. A line group data set named GROUPONE and another line group data set named GROUPTWO that is to be opened idle;
4. A log data set named MSGLOG.

```
Name        Operation    Operands

OPENDISK    OPEN         (DISKREUS,(INOUT),DISKNON,(INOUT))

OPENCKPT    OPEN         (TPCHK,(INOUT))

OPENLINE    OPEN         (GROUPONE,(INOUT),GROUPTWO,(INOUT,IDLE))

OPENLOG     OPEN         (MSGLOG,(OUTPUT))
```

Note that the message queues data sets are opened first and that the checkpoint data set is opened second.

READY Macro Instruction

The READY macro:

- Completes initialization and activation of the MCP,
- Permits "Good Morning" and "Restart in progress" messages to be specified,
- Must be issued between the OPEN macros and the CLOSE macros in the activation and deactivation section of the MCP.

The READY macro completes initialization and activation of the Message Control Program; once READY has executed, the TCAM system is ready for message traffic. One READY macro is specified per MCP. This macro is located between the OPEN macros and the CLOSE macros in the activation and deactivation section of the MCP.

Two optional operands of READY provide the addresses of user-written routines that may build "Good Morning" or "Restart in Progress" messages, and might also alter option fields and other control areas to reflect the fact that a restart has occurred. The exit for the "Good Morning" message is taken for the initial startup of the MCP and for each cold restart; the exit for the "Restart in Progress" message is taken for a warm or a continuation restart (for a discussion of the various types of TCAM startup, see *TCAM Checkpoint/Restart Facility* in *Using TCAM Service Facilities*).

When initial startup or a restart occurs, the appropriate routine is given control for each station defined by a TERMINAL macro, provided that the line group data set containing the line on which the station is located has been opened by means of an OPEN macro. The user routine should save and restore registers. When control passes to the user routine, register 1 contains the address of a two-word parameter list. The first word in the list contains the address of the terminal table entry for the station to which the message generated by the user is to be sent, while the second word contains the address of the option fields for the destination station. The user routine may use this information to build a message tailored to this particular station, if a message is desired, and may also alter fields in the terminal table entry and the option fields for the station to reflect the fact that a restart has occurred (for a warm start or continuation restart, the data in the terminal table entries and the contents of the option fields before closedown or failure are preserved by the checkpoint facility).

The user routine returns to the MCP in register 15 the address of a message to be sent to the station. An all-zero address indicates that no message is to be sent to this station. At the specified address is a one-byte field indicating, in binary form, one more than the number of bytes of data in the message, followed by the text of the message. The maximum length of the message is 255 bytes. If queuing is by terminal, TCAM places the message at the head of the queue for the destination station so that it is the first message sent to that station following startup or restart. The message is handled by the outgoing group of the Message Handler for the destination, and is transmitted like any other message. Since the message is handled by an MH group, it must have a header similar in format to the headers of messages normally handled by the group. The user must construct this header in his exit routine and include it as the first part of his message.

If queuing is by line, the good morning or restart messages for the stations on a line will be placed at the head of the destination queue for that line, and will be sent before any other messages on that queue.

The READY macro has the following format:

Name	*Operation*	*Operands*
[symbol]	READY	[GMMSG=routine] [,RSMSG=routine]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

GMMSG=routine

Function: Specifies the name of a user-written closed subroutine that builds "Good Morning" messages on each line.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.

Notes: If this operand is coded, the routine is given control for the initial startup and for each cold restart.

RSMSG=routine

Function: Specifies the name of a user-written closed subroutine that builds "Restart in Progress" messages.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: If this operand is coded, the routine is given control for a warm restart or a continuation restart.

NOTE: The routines are coded and assembled as part of the MCP in the same manner as the Message Handlers. Their exact location is not important since they are called as closed subroutines. See *User Code in a Message Handler* in the chapter *Designing the Message Handler* for the correct linkages.

Deactivation

Orderly deactivation of the TCAM system involves a number of steps. Incoming and outgoing message traffic must be stopped, and if the status of the system is to be preserved, a checkpoint record must be made. Data sets for any application program using TCAM as its access method must be closed. Finally, the MCP data sets must be closed and control returned to the OS Supervisor.

Types of Closedown

Closedown of a TCAM System may be initiated either by a SYSCLOSE operator command or by an MCPCLOSE macro issued in an application program. These two means of initiating closedown are described in the next two sections. Both the operator command and the macro have an operand that specifies whether a flush closedown or a quick closedown is to be effected. A flush closedown may also be internally initiated when the nonreusable disk threshold is reached.

A *flush closedown* is one in which incoming message traffic on each line ceases after the message being received at the time closedown is ordered has been completed (i.e., the line is not repolled or re-enabled). As soon as incoming message traffic on each line ceases, any eligible outgoing messages that have been queued for stations on that line are sent. (An *eligible* message is a message to a station or application program that is not intercepted; see the description of the HOLD macro.) In this manner, incoming message traffic declines to nothing, while outgoing message traffic continues until all eligible messages have been sent. An environment checkpoint record is taken after all eligible outgoing messages have been sent. This record preserves the status of the MCP and also records the locations on disk of outgoing messages that could not be sent because their destinations were intercepted; after restart, these messages are sent once their destinations are eligible to receive them. This form of termination is known as a *flush closedown* because unsent messages are flushed from the message queues.

When a *quick closedown* is ordered, message traffic stops on each line as soon as transmission of any message currently being sent or received on the line has been completed. Queues of messages to be sent are not flushed but their status is preserved by an environment checkpoint record, and they are sent to their appropriate destinations after restart. (See the discussion of the TCAM checkpoint/restart facility in the chapter *Using TCAM Service Facilities*.)

Deactivating a TCAM System Without Application Programs

If there are no application programs in the TCAM system, a SYSCLOSE operator command entered at an operator control station deactivates the system. The SYSCLOSE command is discussed in the operator control section of the chapter *Using TCAM Service Facilities.*

The SYSCLOSE command specifies either a quick or a flush closedown. When the command is executed, traffic is suspended on each line, as described above.

When all message traffic and TCAM disk operations are complete, control returns to the first instruction following the READY macro in the message control program. This instruction must begin a user-written routine (or branch to a routine) that deactivates the message control program. This MCP deactivation routine must issue CLOSE macro instructions for each open data set in the message control program.

The last TCAM data sets to be closed must be the checkpoint and then the DASD

message queues data sets. This is important, because closing these data sets deactivates the telecommunications system. After the message queues data sets have been closed, no further references can be made to queues, control blocks, the terminal table, invitation lists, etc. The deactivation routine should end with a RETURN macro to end the message control job. (For a sample MCP deactivation routine, see the last section of this chapter.)

Deactivating a TCAM System With Application Programs

When the TCAM system includes application programs, closedown may be effected by an MCPCLOSE macro issued as part of a termination routine in an application program. A recommended procedure is to enter a special closedown message at a station; this message would be directed to each active application program in the system (by specifying the names of the appropriate process entries in the terminal table as its destinations). Each application program might contain a user-written termination routine that would be activated when the message was received. The termination routine might perform the following steps:

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

The user could code the SETEOF macro to execute on his closedown message. When the application program receives the message on which SETEOF has executed, it branches to the address specified by the EODAD= operand of the input DCB macro when the next GET or CHECK macro is issued; at this address the user would have his closedown routine.

When multiple application programs are being closed, an MCPCLOSE macro may be issued in each; the MCPCLOSE macro issued first is the only one to execute.

The MCPCLOSE macro checks to see whether an MCPCLOSE macro has already been issued; if so, the macro does not execute, but places a return code of X'00' in register 15. The first MCPCLOSE macro issued causes all message traffic on TCAM lines to cease, as described above in the discussion of the types of closedown. An operand of MCPCLOSE specifies either a quick or a flush closedown. After all message traffic has ceased, the Message Control Program checks for open application-program data sets; when all such data sets are closed, control passes to the instruction following the READY macro in the MCP. This instruction begins a user-written routine (or branches to a routine) that issues CLOSE macros for each data set opened in the MCP and ends with a system RETURN macro. A sample routine is given in the last section of this chapter.

Instead of using an MCPCLOSE macro, the user may utilize the SYSCLOSE operator command to close a TCAM system having application programs. If any application program data sets are open at the time message traffic ceases, error messages are directed to the system console; the error messages list the open data sets. When these data sets are closed, the system is deactivated.

CLOSE Macro Instruction

The CLOSE macro:

- Is issued in the Message Control Program to deactivate any log data set, line group data set, checkpoint data set, and DASD message queues data set that is open in the MCP,
- Must appear following the READY macro or be branched to from instructions following READY.

The CLOSE macro has the following format:

Name	*Operation*	*Operands*
[symbol]	CLOSE	(dcbname,,,...)

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

(dcbname,,,...)

Function: Specifies the names of the data control blocks for the data sets being closed.
Default: None. This operand is required.
Format: Each *dcbname* must conform to the rules for assembler language symbols, and must correspond to the name specified on the DCB macro for the data set being closed.
Notes: Register notation may be used, in which case the addresses of the data control blocks must previously have been loaded into the general registers specified.

All MCP data sets of the same type (e.g., all line group data sets or all message queues data sets) can be closed with one CLOSE macro by including the names of their data control blocks as operands. If more than one *dcbname* is coded in a CLOSE macro, the names are separated by double commas.

If present, the last TCAM data sets to be closed must be the checkpoint and then the DASD message queues data sets.

Sample MCP Activation and Deactivation Section

The code in Figure 8 represents an activation and deactivation section as it might appear in an MCP. The section consists of an INTRO macro, some user code that tests whether INTRO worked correctly and causes termination of the MCP if INTRO did not execute properly, OPEN macros, a READY macro, and CLOSE macros. The TCAM macros in this section should appear in the order shown.

Since the CPB= operand of the INTRO macro is omitted (and DISK=YES is coded), the TCAM system will issue a WTOR message at INTRO execution time; in his response to this message, the user may specify a value for CPB= and override or supply values for many other INTRO operands.

The section of user code following the INTRO macro is optional, but the user should test the return code in register 15 to determine whether INTRO has executed correctly. If register 15 contains anything other than zero after execution of INTRO, the chances are that the MCP will not work properly.

The OPEN macro in Figure 8 is the same as that used as an example in the previous section. The DASD message queues data sets are opened first, and the checkpoint data set is opened second. (Neither of these data sets is required, but if present they should be opened in the order shown.)

A user-written subroutine may be utilized to perform error checking and correction when a TCAM data set fails to open. The EXLST= operand of the DCB macro for the data set may be coded so that control passes to the user's subroutine whenever the data set fails to open. (For more information on the EXLST= operand, see the descriptions of the DCB macros for the various TCAM data sets.) In this example, the flags set by the OPEN macro are tested for successful completion. Open flags are at an offset of 48 beyond each DCB macro and are set to 16 if the data set is opened correctly. If the open flags are not equal to 16, the abnormal exit is taken.

The READY macro is the last macro of the activation section. RTNA is the name of a user-written closed subroutine that will be activated for the initial startup and each cold restart; RTNB is given control for warm and continuation restarts. Both routines are entered once for each station represented by an entry in the terminal table and located on a line whose line group data set has been opened.

The first CLOSE macro begins the deactivation section. This CLOSE will not be executed until all data sets in TCAM application programs have been closed down, and until all lines have been closed to traffic by means of a SYSCLOSE operator command or an MCPCLOSE macro issued in an application program. The first CLOSE is given control by TCAM once line traffic has ceased. Notice that the DASD message queues data set is closed last, immediately after the checkpoint data set; this practice should be followed when these two data sets are present.

The instructions following CLOSE return control to the OS Supervisor.

```
TCAMINIT INTRO PROGID=MCPONE,            PROGRAM IDENTIFICATION          *
               DISK=YES,                 USING DISK                      *
               CONTROL=OPID,             FOR OPERATOR COMMANDS           *
               KEYLEN=48,                UNIT LENGTH                     *
               LNUNITS=20,               NUMBER OF UNITS                 *
               DLQ=NYC,                  DEAD LETTER QUEUE               *
               USEREG=10,                REGISTERS TO BE SAVED           *
               CPINTVL=1000,             CHECKPOINT INTERVAL             *
               STARTUP=W,                WARM RESTART                    *
               CKREQS=2,                 CHECKPOINT REQUESTS             *
               PASSWRD=VALIDMSG,         PASSWORD FOR APPLICATIONS       *
               CROSSRF=10,               CROSS-REFERENCE ENTRIES         *
               OLTEST=10                 ON-LINE TESTS
         LTR   15,15                     TEST IF INTRO WORKED
         BZ    OPENDISK                  IF SO OPEN DATA SETS
*
NOEXEC   ABEND 123,DUMP                  IF NOT TERMINATE WITH DUMP
*
OPENDISK OPEN  (DISKREUS,(INOUT),DISKNON,(INOUT)) OPEN DISK DATA SETS
         TM    DISKREUS+48,16           TEST IF OPENS WORKED
         BNO   NOEXEC                   NO - TERMINATE
*
         TM    DISKNON+48,16
         BNO   NOEXEC
*
OPENCKPT OPEN  (TPCHK,(INOUT))           OPEN CHECKPOINT DATA SET
         TM    TPCHK+48,16
         BNO   NOEXEC
*
OPENLINE OPEN  (GRCUPONE,(INOUT),GROUPTWO,(INOUT,IDLE)) OPEN LINES
         TM    GROUPONE+48,16
         BNO   NOEXEC
*
         TM    GROUPTWO+48,16
         BNO   NOEXEC
*
OPENLOG  OPEN  (MSGLOG,(OUTPUT))         OPEN MESSAGE LOGGING DATA SET
         TM    MSGLOG+48,16
         BNO   NOEXEC
*
ALLSWELL READY GMMSG=RTNA,RSMSG=RTNB     BEGIN EXECUTION
*
FINISHUP CLOSE (GRCUPONE,,GROUPTWO)      CLOSE LINE GROUP DATA SETS
         CLOSE (MSGLOG)                  CLOSE LOG DATA SET
         CLOSE (TPCHK)                   CLOSE CHECKPOINT DATA SET
         CLOSE (DISKREUS,,DISKNON)       CLOSE DISK DATA SETS
         L     13,4(13)                  PREPARE TO RETURN
         RETURN (14,12),,T               RETURN CONTROL TO OS SUPERVISOR
```

Figure 8. Sample Activation and Deactivation Section

The heart of a Message Control Program consists of the *Message Handlers*, the sets of routines that determine the operations upon messages being received from or sent to remote stations or application programs. A Message Handler is defined by a sequence of TCAM macro instructions and is constructed to handle messages for a particular line group, or for several line groups that have similar characteristics.

The purpose of a Message Handler (MH) is to define the macro-introduced routines that:

1. Examine and process control information in message headers;
2. Perform necessary functions in preparing message segments for forwarding to their destinations, which may be stations or application programs.

There are two kinds of macro instructions that may be included in a Message Handler, functional and delimiter macro instructions. The *functional macros* perform the specific operations required for messages directed to the Message Handler. *Delimiter macros* classify and identify sequences of functional macro instructions and direct control to the appropriate sequence (some delimiter macros have limited functional capabilities).

Design of a Message Handler consists of selecting certain TCAM macro instructions described in this chapter, and writing them in a particular sequence, according to the requirements of the application and the characteristics of the lines. Careful analysis must be made of such considerations as the type of station and the type of line in use, the processing requirements of different types of messages, and the format of the message headers to be handled.

Before discussing the Message Handler and its parts, we shall briefly consider the format of the TCAM message and its message header.

Message Format

A message may consist of two parts, the header portion and the text portion. The *header* portion contains control information for the message, such as:

- One or more destination codes,
- The code name for the originating station,
- The number of the message relative to the numbers of the previous messages received from that station (input sequence number),
- A message-type indicator,
- Various other fields containing control indicators.

The *text* portion of a message consists of information of concern to the party ultimately receiving the message, either a station or an application program.

Depending on the application, messages may consist of a header only, text only, or header and text. A header-only message may utilize a message-type indicator to route the message to an application program and, possibly, obtain a standard response. If all messages go to only one application program, such as a file-update program, the header may be omitted.

The determination of what part of the message is the header and what part is text is up to the user.

Depending on the type of work unit with which he is dealing, the user must specify appropriate characters for control purposes. The types of work units are defined as follows:

- A *block* is that portion of a message terminated by an EOB or ETB control character or, if this is the last block in the message, by an ETX or EOT control character. A *subblock* is that portion of a BSC message terminated by an ITB.
- A *segment* is that portion of a message contained in a single buffer. The size of the buffer is specified by the user for each line group and application program.

- A *record* for an application program is, most often, that portion of a message terminated by a format character (ESC, NL, TAB, CR, or LF), or, less often, a message portion terminated by a character specified by the data operand of the MSGEDIT macro (see the description of this macro).
- A *message* is a unit of data terminated by an EOT or ETX control character or, if the CONV= operand of the STARTMH macro is coded CONV=YES, by an ETB or EOB control character (see the description of the STARTMH macro).

The Message Header

Operations on the fields of the message header are the primary function of the Message Handlers in the Message Control Program. The length and format of the header and the information it contains depend solely on the requirements of the application and the user's preferences. The length may be a few characters or many characters. A header may occupy more than one buffer. However, the entire header of a message must be contained within the first block of the message. EOBs may not be embedded within a header.

The format of the message header dictates the arrangement of the appropriate Message Handler macros. The control characters used and the sequence of fields within the header must be predetermined so that the Message Control Program can be properly coded.

Destination codes in the message header identify the stations or application programs to which the message is to be routed. The message-type indicator can identify a header that is to be processed in a special manner. By inserting certain macro instructions in the Message Control Program, the user can insert in the header such data as the date and time it is sent, and the output sequence number.

There are many possible variations for the format of a message header. The sample formats shown in Figures 9 and 10 are included simply for illustrative purposes.

The format shown in Figure 9 could be used in a message switching application. This figure shows how an incoming message might look just before it comes into main storage.

In this example, the EBCDIC blank character (here denoted by the symbol ℔ serves as a delimiter for each header field. This is not always the case, however; some MH macros operating on the header do not look for field delimiters, but consider a certain number of characters or a certain sequence of characters to be a header field. To determine what constitutes a header field for any particular macro, the user should consult the description of that macro in the section *Functional Macro Instructions*.

Byte 0 contains a machine end-of-address (EOA) character inserted by the station. When the message is transmitted, this character signals the end of nonrecorded machine control characters (such as addressing characters and the machine EOA itself) and the beginning of data characters. Depending upon how the LC= operand of the STARTMH macro of the Message Handler is coded, TCAM may remove the machine-control characters and the machine EOA before placing the message in main storage. The 192 in bytes 1 through 3 is the input sequence number. Bytes 5 through 7 contain the code for the terminal that originated the message. Bytes 9 through 11 and 13 through 15 contain destination codes specifying the stations to which the message is to be sent. In this example, the semicolon in byte 17 has been designated by the user as the program EOA character. Since some of the messages in this application contain multiple destination codes, the destination delimiter character must follow the last destination code (for more on the program EOA character, see the description of the FORWARD macro). Byte 19 contains characters specifying the priority of the message. The remaining portion of the message is text and is followed by the EOT character (which must be inserted by the station operator to indicate the end of the message).

If LC=OUT is coded in the STARTMH macro, all control characters (including the machine EOA) are deleted after the message is placed in buffers; thus, the user must allow room for these characters in his buffers. If he is going to insert time-received, date-received, and output-sequence information into his message header, the user must specify the number of bytes to be reserved in his input buffer for this information by the RESERVE= operand of the line group DCB macro. (The user may also insert data into his message by the MSGEDIT macro; no buffer space need be reserved for data inserted by MSGEDIT.)

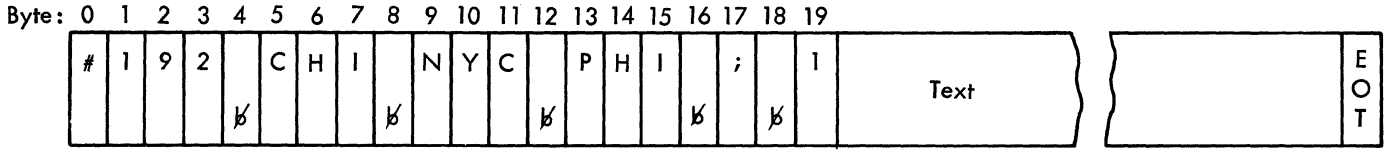


Figure 9. Sample Format for an Incoming Message

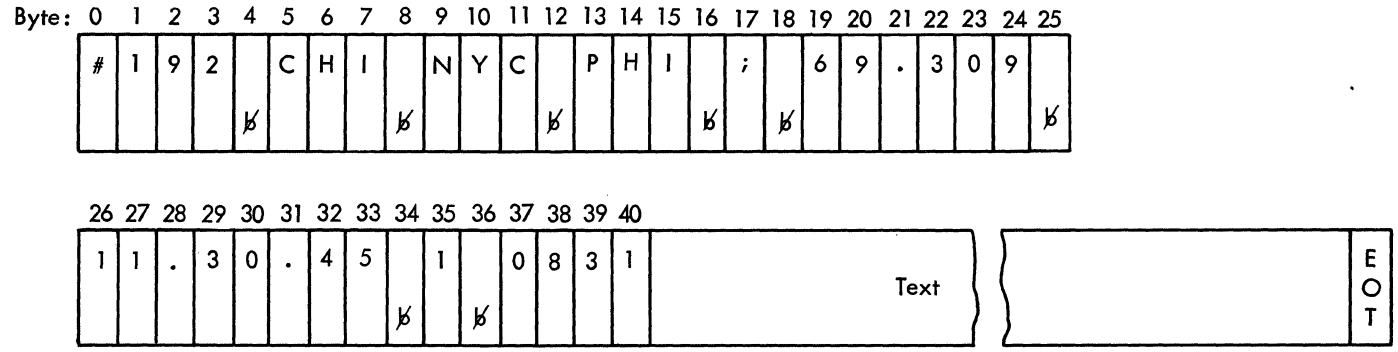


Figure 10. Sample Format for an Outgoing Message

Figure 10 shows how the message would look as it was transmitted to a destination station. In this example, the Message Control Program inserted time-received and date-received information in the header. The time-received information in bytes 26 through 33 indicates that the message was received at 11 hours, 30 minutes, and 45 seconds on the date specified in bytes 19 through 24, which is November 5, 1969. Insertion of this information moved the priority data to byte 35. The message is then queued by priority on the message queue for the destination station. When the message reentered main storage prior to transmission to the destination stations, the Message Control program placed a blank followed by the output sequence number in bytes 36 through 40 of the header. TCAM sends a series of control characters (ending with the machine EOA) before sending the message to its destination; TCAM supplies an EOT character when the MSGFORM macro is coded.

TCAM, with its complete set of header-processing routines and associated macro instructions, allows the user to indicate the header-processing functions he wants by including the appropriate macro instructions in the Message Control Program. These functions and the relationship of the message header format to the design of the Message Control Program are discussed later in this chapter.

Structure of the Message Handler

A Message Handler is divided into two main groups of macro instructions, the incoming group and the outgoing group. The incoming group of a Message Handler is designed to handle all messages that arrive for handling by the Message Control Program; these messages may originate from any of the lines, line groups, or application programs that are assigned to have their messages handled by the Message Handler. The outgoing group handles messages being sent from the Message Control Program to any of the lines, line groups, or application programs.

Each of the two groups of a Message Handler may be divided into three kinds of subgroups. The incoming group has the following possible subgroups:

- Inheader subgroups, which handle only those incoming message segments that include all or part of a message header,

- Inbuffer subgroups, which process *all* incoming message segments, and
- Inmessage subgroups, which are executed after a complete message has arrived at the CPU.

The outgoing group also has three possible subgroups:

- Outheader subgroups, which handle only those outgoing message segments that include all or part of a message header,
- Outbuffer subgroups, which process all outgoing message segments, and
- Outmessage subgroups, which are executed after a complete message has been sent.

The following list presents an overview of MH organization, and describes typical functions performed in each type of subgroup. The words in parentheses are the names of the MH macros that perform the functions described.

MACRO OR SUBGROUP	*FUNCTION*
STARTMH	Determines whether this is an incoming or an outgoing message, and routes it to the appropriate MH group.
Inheader Subgroup	Operates on an incoming message-header; does such things as: • Determining whether the message should be translated to EBCDIC (CODE) • Determining the message origin and destination (ORIGIN, FORWARD) • Checking the incoming sequence number (SEQUENCE) • Determining the message priority (PRIORITY) • Editing header fields (MSGEDIT)
Inbuffer Subgroup	Operates on each segment of an incoming message; does such things as: • Counting incoming message segments (COUNTER) • Text editing (MSGEDIT) • Checking the length of incoming messages and terminating reception for messages that are too long (CUTOFF)
Inmessage Subgroup	Specifies actions to be taken after the entire message has been received, such as • Logging the message (LOG) • Canceling the message (CANCELMG) • Returning error messages to the originating station (ERRORMSG, MSGGEN)
Outheader Subgroup	Operates on an outgoing message header; does such things as: • Inserting the date and time into the outgoing header (DATETIME) • Assigning an outgoing sequence number to the message and inserting it in the header (SEQUENCE) • Editing header fields (MSGEDIT) • Determining whether the message should be translated to line code (CODE) • Determining whether line-control characters should be inserted into the outgoing message (MSGFORM)
Outbuffer Subgroup	Operates on each segment of an outgoing message; does such things as: • Counting outgoing message segments (COUNTER) • Text editing (MSGEDIT)

Outmessage Subgroup — Specifies actions to be taken after the entire message has been sent, such as

- Logging the message (LOG)
- Sending an error message to the destination station (ERRORMSG, MSGGEN)
- Causing future messages to the destination station to be held up (because, perhaps, the station is inoperative) (HOLD)

More than one subgroup of a particular kind may be included within a group to accommodate variations in handling that may be required by various kinds of messages (see *Variable Processing within a Message Handler* in this chapter).

Delimiter macros identify the beginning and end of the various MH groups and subgroups. The STARTMH macro identifies the beginning of an MH. INHDR, INBUF, and INMSG respectively identify the beginning of the inheader, inbuffer, and inmessage subgroups of the incoming group. INEND identifies the end of the incoming group. OUTHDR, OUTBUF, and OUTMSG respectively identify the beginnings of the outheader, outbuffer, and outmessage subgroups of the outgoing group, while OUTEND identifies the end of this group. The delimiter macros are discussed in detail later in this chapter.

A minimum Message Handler consists of a STARTMH macro and either an incoming group or an outgoing group (either group may be omitted, as when the incoming group is omitted for an output-only line). If the outgoing group is omitted, the OUTEND macro must be coded to preserve addressability.

The incoming group must precede the outgoing group if both are included in an MH.

The following rules govern the arrangement of subgroups within a group:

1. If there is an incoming group, an inheader subgroup is required as the first subgroup. All other subgroups of the incoming groups are optional.
2. The first inheader subgroup in an incoming group may be followed by any combination of inheader and inbuffer subgroups.
3. Inmessage subgroups, if present, must be the last subgroups in the incoming group.
4. Any of the three types of subgroups for the outgoing group may appear as the first subgroup in the group. However, if an outmessage subgroup is the first subgroup, no outheader or outbuffer may appear in the group.
5. Outheader and outbuffer subgroups may appear in the outgoing group in any order (i.e., either subgroup may appear first and each may be specified more than once to accommodate variations in handling required by different types of messages).
6. Outmessage subgroups, if present, must be the last subgroups in the outgoing group.

The sample Message Handlers in this chapter illustrate some of the ways in which subgroups may be arranged.

The presence or absence of particular groups and subgroups within a given Message Handler depends upon the requirements of the user. Figure 11 summarizes the MH macros that may appear within a given subgroup. The user should familiarize himself with the functions of the macros shown in Figure 11 and decide which of these functions to incorporate into his Message Handler. His choice of functions will determine which subgroups will be present in his MH. For example, if he decides he needs the function provided by the CANCELMG macro in his MH, then he will require an inmessage subgroup. Some macros (CODE, COUNTER, LOG, for example) may appear in more than one kind of subgroup, but their functions vary according to the kind of subgroup in which they appear.

Inheader Subgroup	CHECKPT CODE COUNTER DATETIME FORWARD INITIATE LOCK LOCOPT LOG MSGEDIT	MSGLIMIT MSGTYPE ORIGIN PATH PRIORITY SEQUENCE SETSCAN TERRSET UNLOCK
Inbuffer Subgroup	CHECKPT CODE COUNTER CUTOFF LOG	LOCOPT MSGEDIT PATH TERRSET
Inmessage Subgroup	CANCELMG CHECKPT ERRORMSG HOLD	LOG MSGGEN REDIRECT
Outheader Subgroup	CHECKPT CODE COUNTER DATETIME LOG MSGEDIT MSGFORM MSGLIMIT	LOCOPT MSGTYPE PATH SCREEN SETEOF SETSCAN SEQUENCE TERRSET
Outbuffer Subgroup	CHECKPT CODE COUNTER LOCOPT	LOG MSGEDIT PATH TERRSET
Outmessage Subgroup	CHECKPT ERRORMSG HOLD	LOG MSGGEN REDIRECT

Figure 11. MH Subgroups and Macros

Selecting Message-Handler Functions

Functional macro instructions perform the specific operations required for message segments being handled by the various subgroups of a Message Handler. Message segments are directed to the appropriate subgroup by the delimiter macros; the functional macros of the subgroup are then executed in the order in which they are specified within the subgroup. Functions provided by an MH include:

- Message editing (insertion of date, time, and sequence number, insertion or removal of characters or character strings).
- Validity checking (verification of source and destination codes and of sequence numbers in incoming message headers).
- Routing messages to various destinations or alternate destinations, possibly by priority.
- Maintaining counts and logs for message traffic on a line.
- Error checking and handling (checking for errors in transmission and taking corrective action).
- System control (interrogating or modifying activity on a system, line, or station basis, or specifying properties or limitations for messages).
- Function selection (permitting dynamic selection of the functions to be performed on messages).

The table below shows the various macros used to specify these functions. A brief description of the Message Handler functions provided by TCAM is given here. A complete description of these functions is found in the discussions of the individual macros in the *Functional Macro Instructions* section of this chapter.

MH Functions and Macros Defining the Functions

Message Editing	CODE, DATETIME, MSGEDIT, MSGFORM, SEQUENCE
Validity Checking	FORWARD, ORIGIN, SEQUENCE
Routing	FORWARD, INITIATE, MSGGEN, PRIORITY, REDIRECT
Record Keeping	CHECKPT, COUNTER, LOG
Error Handling	CANCELMG, CUTOFF, ERRORMSG, HOLD, MSGGEN, REDIRECT, TERRSET
System Control	CUTOFF, HOLD, INITIATE, LOCK, LOCOPT, SCREEN, SETEOF, MSGLIMIT, UNLOCK
Function Selection	MSGTYPE, PATH, SETSCAN

Message Editing

Five TCAM macro instructions –CODE, DATETIME, SEQUENCE, MSGEDIT, and MSGFORM – provide editing facilities.

CODE, when specified in the inheader or inbuffer subgroup, translates the data in the buffer from the line code to EBCDIC, using a specified translation table. When specified in the outheader or outbuffer subgroup, translation is from EBCDIC to the line code.

DATETIME inserts the date and the time at which the message is received by (or sent by) the MCP in the header.

SEQUENCE, when specified in an outheader subgroup, inserts an output sequence number in the messages that are sent to a destination.

MSGEDIT inserts or deletes a character or character string in the message.

MSGFORM, when specified in an outheader subgroup, inserts blocking characters into outgoing messages, thereby dividing the messages into logical blocks of data.

Validity Checking

Three TCAM macro instructions – FORWARD, ORIGIN, and SEQUENCE – provide validity checking of fields in the message header.

FORWARD verifies that the station codes specified as destinations in the message header are valid destinations in the sytem.

ORIGIN determines the station that entered a message by checking the origin field in the message header for the symbolic name of the station. The origin field is that field of the header allotted to contain the name of the originating station during the design of the header and the Message Handler.

SEQUENCE verifies that the input sequence number included in the message header by the station operator is valid; that is, that the number is one greater than the sequence number of the previous message from that station. To perform this function, SEQUENCE is included in an inheader subgroup.

Message Routing

Five TCAM macro instructions – FORWARD, INITIATE, MSGGEN, PRIORITY, and REDIRECT – perform functions related to the routing of messages to a particular destination.

FORWARD routes messages to the destinations specified in the message headers or to the destinations specified by the FORWARD macro.

INITIATE routes segments of messages to their destination as soon as they are received, without waiting for the whole message to arrive.

MSGGEN generates a special response message and routes it immediately to either the originating or the destination station. The response message bypasses normal message handling, queuing, logging, and buffering functions.

PRIORITY routes messages to their destinations according to priority levels specified either in the message header or by the PRIORITY macro.

REDIRECT queues a message for an additional or alternate destination under certain error conditions, or unconditionally.

Record Keeping

Two TCAM macro instructions – COUNTER and LOG – enable records to be kept of the flow of messages in the system.

COUNTER keeps a count of incoming or outgoing message segments or complete messages, depending on the subgroup in which the macro is issued.

LOG places copies of segments or messages passing through the system on a sequential medium, such as magnetic tape.

Error Handling

Seven TCAM macro instructions – CANCELMG, CUTOFF, ERRORMSG, HOLD, MSGGEN, REDIRECT and TERRSET – provide facilities for the detecting and handling of errors. These macro instructions test for error conditions arising during transmission and handling of the message, and take action accordingly. The seven macros are used in conjunction with a message error record, which is assigned to each message as the message is handled. The meaning of each of the bits in the message error record is explained in *Appendix B.* Some error-handling macros (CUTOFF, TERRSET) set bits in the message error record; the rest test the bits in the message error record.

A five-byte bit configuration (called a *mask*) is specified in some error-handling macros. When the macro is executed, the mask is compared to the message error record assigned to the message. If a 1 is detected in any bit position of both the mask and the message error record, the functions specified by the macro are performed. A 0 is specified in a mask bit position when the error condition represented by the corresponding position in the message error record is to be ignored. (An operand of the error-handling macro may specify that the macro is to be executed only if *all* bits specified in the mask are on in the message error record.)

The function specified by an error-handling macro may also be performed unconditionally (that is, for all messages or message blocks, independent of the setting of the message error record) by either specifying a mask consisting entirely of zeros or not specifying a mask at all.

The requirements of the application must be analyzed to determine which errors or conditions must be detected, and which can reasonably be ignored without degrading the performance of the system. The seven error-handling macro instructions provide varying methods by which corrective or control functions can be initiated when an error has been detected.

CANCELMG cancels a message if a specified error has occurred.

CUTOFF checks for incoming buffers filled with identical characters (an indication of station malfunction). In such a case, the appropriate bit is set in the message error record. CUTOFF also specifies the maximum number of characters allowed in a message; if the maximum is exceeded, reception is terminated and an error bit is set.

ERRORMSG and MSGGEN send a specified message if a specified error has occurred.

TERRSET sets a bit in the message error record to indicate, at the discretion of the user, that a user-defined error has occurred.

HOLD suppresses the sending of messages to a station when an error specified by the mask has been detected; it is usually used to withhold transmission to an inoperative station.

REDIRECT sends a message to an additional or an alternate destination if a specified error has been detected. This function normally handles messages that cannot be sent to their intended destinations.

The FORWARD, ORIGIN, and SEQUENCE macro instructions set bits on in the message error record when they detect an invalid destination, origin, and sequence number, respectively.

System Control

Nine TCAM macro instructions – CUTOFF, HOLD, INITIATE, LOCK, LOCOPT, MSGLIMIT, SCREEN, SETEOF, and UNLOCK – provide facilities for modifying the telecommunications system or for providing dynamic control over the functions of the system.

CUTOFF terminates transmission of an excessively long message.

HOLD stops transmission of messages to a station known to be inoperative or unattended for a period of time making transmission undesirable.

INITIATE sends message segments to their first destination before the entire message has been received and enqueued.

LOCK maintains the connection between a station and an application program for the duration of a message and its response. This facility is used for fastest response during inquiry applications.

LOCOPT provides access to fields of the option table, permitting examination and modification of the contents of the fields.

MSGLIMIT limits the number of messages sent to or received from a station during a single transmission sequence.

SCREEN modifies the WRITE operation for terminals with display screens.

SETEOF indicates an end-of-file message for an application program.

UNLOCK removes a station from the LOCK condition.

Function Modification

Three TCAM macro instructions – MSGTYPE, PATH, and SETSCAN – permit modification of the functions of the Message Handler. The first two macros provide for variations in the processing provided by the MH for different types of messages. SETSCAN permits modification of the scan pointer setting (discussed below) to allow processing of a field in the header in some order other than the normal sequential order.

Functions Provided by Delimiter Macros

STARTMH provides end-of-block checking for hardware and logical errors and takes appropriate user-specified action when such errors are detected. These errors cause bits to be set in the message error record. STARTMH also provides a capability for the automatic deletion of certain line-control characters, so that the user need not concern himself with them. Of the remaining delimiters, INHDR, INBUF, INMSG, OUTHDR, OUTBUF, and OUTMSG provide path-switching facilities. Only the INEND and OUTEND delimiter macros have no functional capabilities.

Order of Macro Specification

Functional macros relating to an entire message segment (i.e., MSGEDIT, LOG) may appear at any point within the subgroup in which they are used. Those relating to specific header fields (i.e., ORIGIN, DATETIME) should appear in the same order within the inheader or outheader subgroup as the header fields appear within the header. In planning a format for message headers, the user may arrange the various header fields in any desired order within the header; the corresponding macros that act on those fields must be placed within the subgroup in the same order. These order-dependent macros involve either:

- Inserting a new field in the message header (e.g., DATETIME).
- Making a decision at some point during header processing (e.g., MSGTYPE), or
- Using a TCAM scanning routine to determine the contents of a specific field (e.g., ORIGIN).

Order is important with the macros relating to a specific header field because these functions rely on a pointer (known as the *scan pointer*), which must refer to the proper header field when the macro that acts upon that field is given control. The use of the scan pointer is described below.

The Scan Pointer

In handling a buffer, TCAM maintains a pointer to the current field in the message header. Some macro instructions specified by the user 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. The user must be aware of the positioning of the scan pointer as he designs his Message Handler.

There are basically two types of TCAM macro instructions that move the scan pointer automatically, without intervention by the user. Examples can be found in Figure 12.

1. Certain macros (e.g., SETSCAN, FORWARD) move the scan pointer until a user-specified character string is found. After these macros have completed execution, the scan pointer is positioned on the last character acted upon. In this case, the next character string is assumed to be the next field to be looked at by the next field-dependent macro in the Message Handler.
2. Other macro instructions move the scan pointer a certain number of characters. There are three ways this number is determined.

   a. With certain macros (FORWARD, ORIGIN, SETSCAN), the user may specify explicitly a number of nonblank characters to be considered as the next field. When these macros have completed, the scan pointer is positioned to the last character that satisfies the count.

   Example:
   FORWARD DEST=3.
   If a destination field in a header reads NYBC, the scan pointer points to the C.

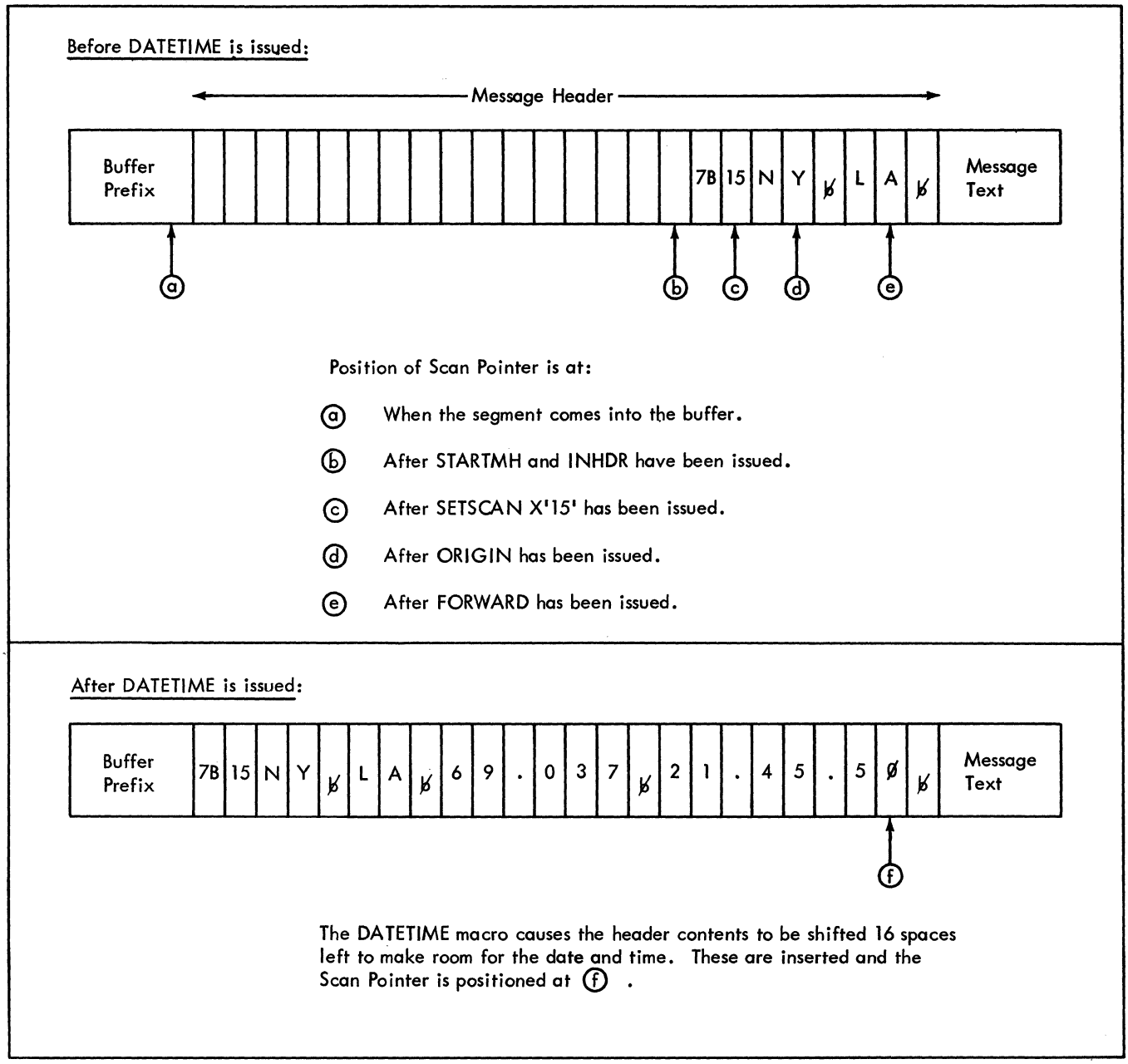


Figure 12. Scan Pointer Movement

b. Some macro instructions may be concerned with fields of varying length, such as FORWARD and ORIGIN. The scan pointer is moved past any blanks that might precede the field. The field is then scanned for a blank delimiter. When such a macro is completed, the scan pointer points to the character immediately preceding the blank delimiter that designates the end of the field.

c. Some macros are concerned with fields whose length is specified implicitly by coding the contents of the field in the macro itself. SETSCAN may search for a field with contents matching those specified in the macro; the position of the scan pointer after the macro executes is described in the discussion of the macro. The INITIATE, LOCK, MSGTYPE, PATH, PRIORITY, SCREEN, SETEOF, and UNLOCK macros have an optional *conchars* operand; when this operand is specified, the scan pointer is moved past any blanks preceding the next field in the message, and the contents of the field are compared with the *conchars* string. If the field and the *conchars* string match, the scan pointer points to the last character of the field; otherwise, it is returned to the position it occupied before the comparison was made.

When a message segment is received for processing in an incoming group of an MH, the space reserved for expansion by the RESERVE= operand of the line group DCB macro is moved to the front of the segment and the scan pointer is positioned to the last reserved byte. If no reserve bytes were specified, the scan pointer points to the last byte of the buffer prefix. The buffer is then examined for the presence of a machine EOA sequence. If such a sequence is found, the scan pointer is moved to the last byte of the sequence. However, the polling character used when the Auto Poll feature is active is not considered to be part of the machine EOA sequence.

For an incoming message, the scan pointer is not necessarily at the end of the header when the inheader subgroup finishes executing. The header may have additional fields that are to be operated upon by an outheader subgroup when the message is removed from its destination queue. In this case, when the message is removed from the queue, STARTMH and OUTHDR reposition the scan pointer to the last remaining reserve byte or, if there are no more unused reserve bytes, to the last byte of the prefix or, if there is a machine EOA sequence to the last byte of the EOA. If he wishes to use an outheader subgroup to process the remaining fields of his header, the user may use the SETSCAN macro to reset the scan pointer to the last byte of the last field processed by the inheader subgroup.

Macro instructions in an MH should be placed in the same order within a subgroup as the fields of the header on which they act. The scan pointer controls access to these fields, progressing across the header from left to right as the various macro instructions are executed. The user may use the scan pointer (via the SETSCAN macro) in his own routines to perform additional header analysis. However, he must take the responsibility of positioning the scan pointer to its proper position before executing the next TCAM macro.

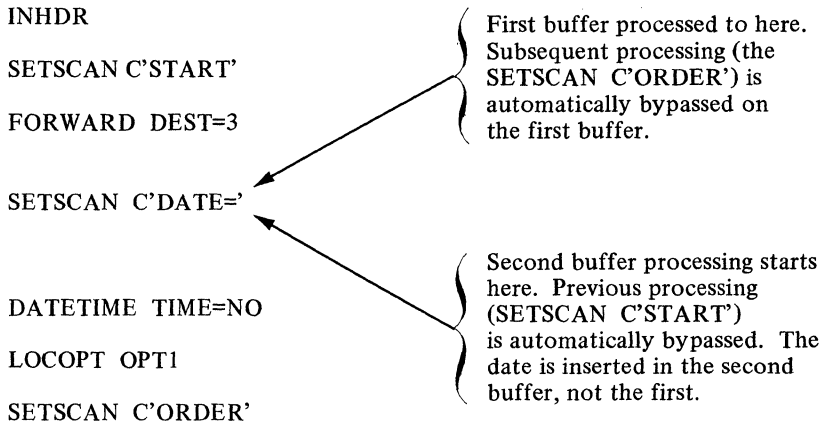
A few TCAM macro instructions perform across buffer boundaries, i.e., they are able to process a field that is partly in one buffer and partly in the next buffer. SETSCAN used to skip to a specified character string is an example.

To perform cross-buffer execution, TCAM upon detecting the fact that the field is incomplete in the current buffer, saves the part of the field that appears in the current buffer in a TCAM internal control block. The address of the parameter list for the function is also saved in this block, along with as many of the user's registers as are specified in the USEREG= operand of the INTRO macro. The scan pointer is then set to point beyond the end of the data in the current buffer. This prevents further processing of data in the buffer by subsequent Message Handler macros. TCAM is able to recognize, on entry, when the scan pointer is beyond the end of data and to discontinue the requested function.

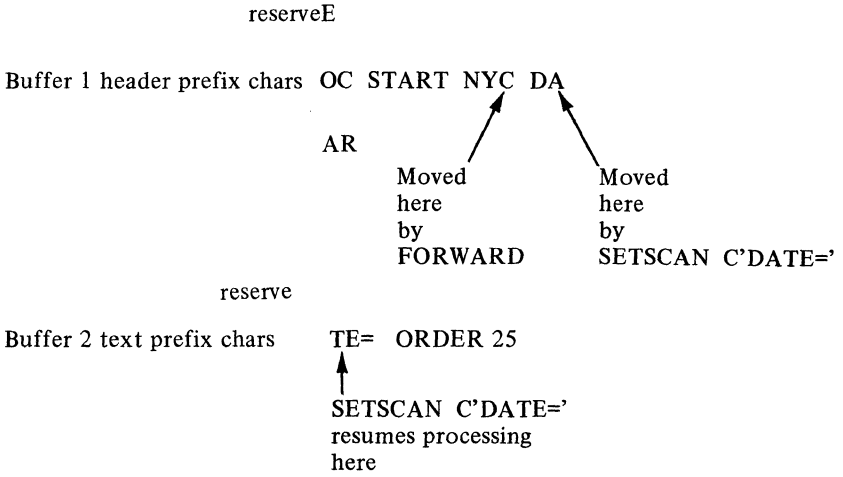
When the next buffer of the message is received by the MH, STARTMH detects the presence of the parameter list address in the control block, which causes STARTMH to exit to the function that failed to complete, rather than beginning to process the first Message Handler macro as on initial entry.

The function thus reentered recognizes that entry is for completion of a previously initiated operation. The data accessed from the previous buffer is recovered from the control block, and processing resumes at the beginning of data in the new buffer. The return point to the MH from the reentered function is calculated from the parameter list address, and sequential processing resumes at that point.

The following example shows the sequence of events in a cross-buffer situation.



.
.
.
.



Not all TCAM Message Handler functions can perform cross-buffer processing. For example, the MSGTYPE macro defaults to an unsuccessful comparison (no match found) unless a string long enough to match the comparison string is found.

Not all MH functions are concerned with the position of the scan pointer. In a cross-buffer situation such as the one illustrated above, the LOCOPT macro executes for both buffer 1 and buffer 2. If such processing is to be performed only once, the macro in question should be so located in the MH that only one buffer will pass through its code. This may be done by:

a. Placing the function early enough in the MH that the first buffer (hence, only the first) passes through it,

b. Testing the return codes of functions that move the scan pointer and branching accordingly, or

c. Insuring that no header extends beyond one buffer.

Message Flow Through a Message Handler

Figures 13 and 14 illustrate the overall flow of a message through Message Handlers written for two representative TCAM applications. After briefly considering the overall flow, the path of a message within a single incoming or outgoing group is described.

Figure 13 illustrates the flow through a single MH of a message to be switched from one station to another that requires no processing by an application program. The incoming message is routed by STARTMH to the incoming group of the MH assigned to the line (by the MH= operand of the DCB macro for the line group in which the line is included). After being processed by the incoming group, the message is placed on the destination queue for the station to which the message is to be routed. This queue may be on a direct-access storage device, or it may be in main storage. TCAM obtains messages from the destination queue on a first-ended first-out basis within priority groups. STARTMH routes the message from the destination queue to the outgoing group of the MH assigned to the line on which the destination station is located. After being handled by the outgoing group, the message is transmitted to the destination station.

Figure 14 illustrates the more complicated message flow for a message that is received, routed to an application program, and then transmitted to a destination station. The message is processed first by the incoming group of the MH handling messages for this line, then placed on the destination queue for the application program (this queue is created by a TPROCESS macro). The outgoing group created especially for the application program and assigned to it by the MH= operand of a PCB macro processes the message when it is removed from the destination queue; the message is then placed on the *read-ahead queue*, a special queue accessed by GET or READ macros issued in the application program. After being processed by the application program, the message is returned to a process queue by PUT or WRITE macros and is handled by the incoming group of the MH assigned to the application program by the MH= operand of the PCB macro for the application program. The message is routed by this incoming group to the destination queue for the station that is to accept the message. It is then handled by the outgoing group of the MH assigned to the line and transmitted to the destination station.

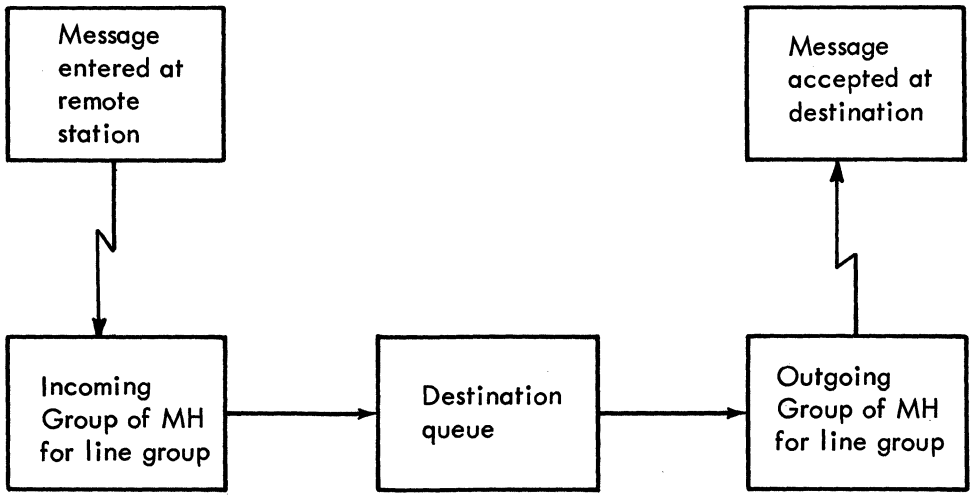


Figure 13. Message Flow for a Switched Message

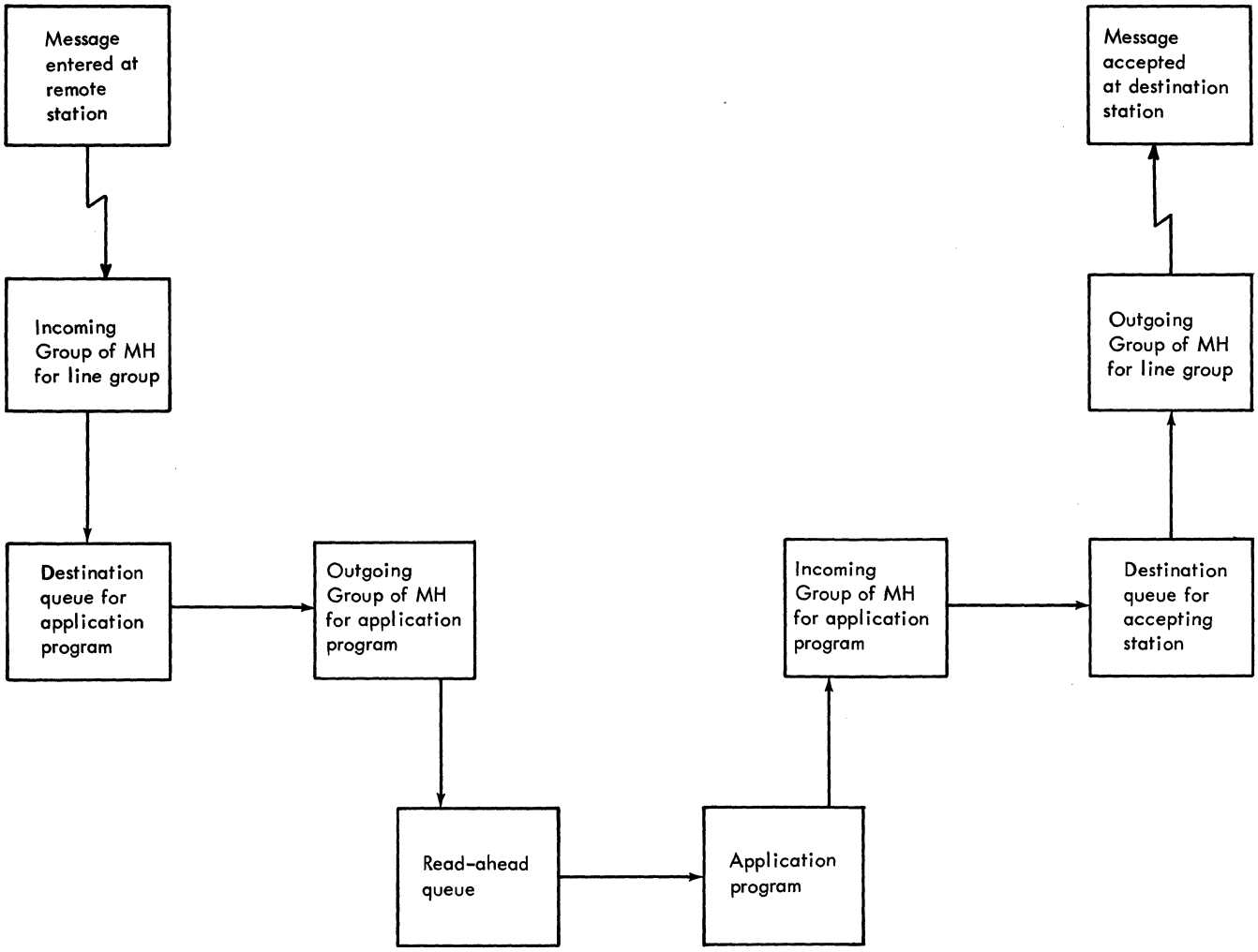


Figure 14. Message Flow for a Message that is Processed by an Application Program

In Figure 14, two incoming and two outgoing groups are used in handling the message. One incoming and one outgoing group are assigned to the line, and one of each group is assigned to the application program. The user might provide these groups by designing one MH for his line and another for his application program; or he might design a single MH and assign it to both the line and the application program. This single MH would have some subgroups that would be executed only for messages coming in from or going to a station on the line, and other subgroups that would be executed only for messages being sent to or received from an application program. For a description of how this selective execution is accomplished, see *Variable Processing within a Message Handler* in this chapter.

For a more thorough description of the flow of a message through a TCAM system, see *Message Flow within the System* in the *TCAM Concepts and Facilities* publication.

Message Flow within an MH Group

That portion of a message contained within one main-storage buffer is called a *message segment.* When a message segment arrives for processing by a Message Handler, STARTMH determines whether the segment is part of an incoming or outgoing message and routes it to the incoming or outgoing group, as appropriate. STARTMH also determines whether the segment contains part of a multiple-buffer header. A *multiple-buffer header* is a message header that occupies more than one buffer. Message segments containing part of a multiple-buffer header go through the inheader and outheader subgroups in a special manner, described below.

The flow through an MH assigned to a line group of a message that does not have a multiple-buffer header is illustrated in Figure 15. After a segment has been routed to an incoming or outgoing group, the INHDR or OUTHDR macro determines whether this is the first segment of a message, or a segment other than the first. Only the first segment of a message not having a multiple-buffer header is routed to the inheader or outheader subgroup (if present). All segments (including the first) are normally processed by the inbuffer or outbuffer subgroups present in the group handling the message. The macros in the inheader, outheader, inbuffer, and outbuffer subgroups are executed on a segment-by-segment basis, while those in the inmessage and outmessage subgroups are not executed until the entire message has been handled by the other subgroups. The inmessage subgroup is executed when the last message segment reaches the inmessage delimiter. The outmessage subgroup is not executed until after the entire message has been transmitted to the destination station or sent to the application program.

In Figure 15 there are only three subgroups per group, and it is assumed that all subgroups are involved in handling the message. Since some subgroups are optional, a group may have fewer than three subgroups. On the other hand a group may have many more than three subgroups, because more than one subgroup of a given type may be included in a group. Moreover, not all subgroups included in a group need be involved when the group handles a particular message; TCAM provides selective execution of subgroups according to the setting of a path switch. This variable-processing capability is discussed later in this chapter.

Multiple-Buffer Header Handling

Figure 16 illustrates the flow through an MH assigned to a line group of a two-segment message having a multiple-buffer header. The main difference between this type of flow and that described above for a message not having a multiple-buffer header is the way in which the inheader and outheader subgroups are executed.

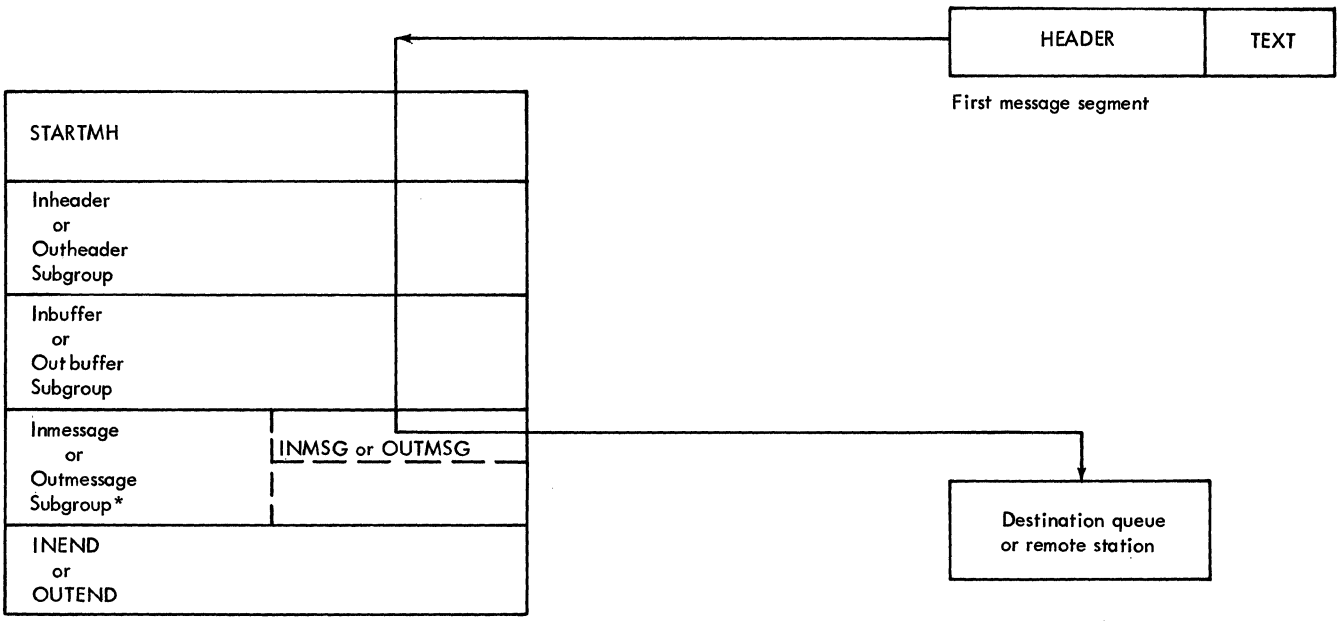
The first segment of a message having a multiple-buffer header consists entirely of header information. This first segment does not go through the entire inheader or outheader subgroup. Once the last field in this segment has been processed by field-dependent instructions in the inheader or outheader subgroup (i.e., once the scan pointer has advanced to the end of the buffer), TCAM saves the address of the next (unexecuted) inheader or outheader instruction and also saves the contents of all registers specified by the USEREG= operand of the INTRO macro.

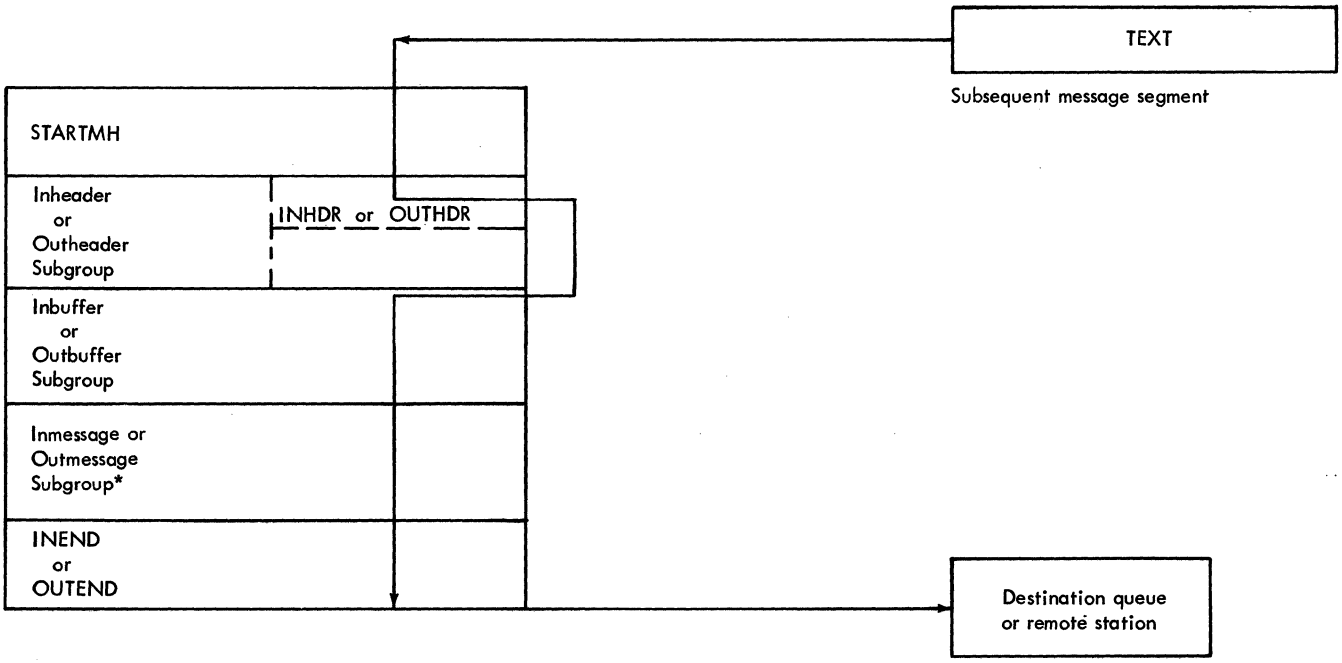
The first segment continues through the inheader or outheader subgroup, but only those macros that do not depend on the location of the scan pointer or upon certain data being in the buffer are executed for it. Among such macros are CHECKPT, CODE, COUNTER, LOCOPT, LOG, MSGFORM, MSGLIMIT, and TERRSET. The INITIATE, LOCK, MSGTYPE, PATH, SCREEN, SETEOF, and UNLOCK macros execute if the *conchars* operand is not coded for them. The FORWARD macro executes if the destination is specified in the macro, rather than in the message header. The PRIORITY macro executes if the priority level is specified in the macro and no *conchars* operand is coded.

In Figure 16, a dotted flow line through the inheader/outheader section indicates that the scan pointer has reached the end of the first header segment, and only those macros listed above are being executed for it.

When the second segment is ready for handling, STARTMH routes it directly to the inheader or outheader instruction whose address was saved, rather than to the INHDR or OUTHDR macro at the beginning of the subgroup. (At this time, TCAM also restores the contents of the registers specified by the USEREG= operand of the INTRO macro.)

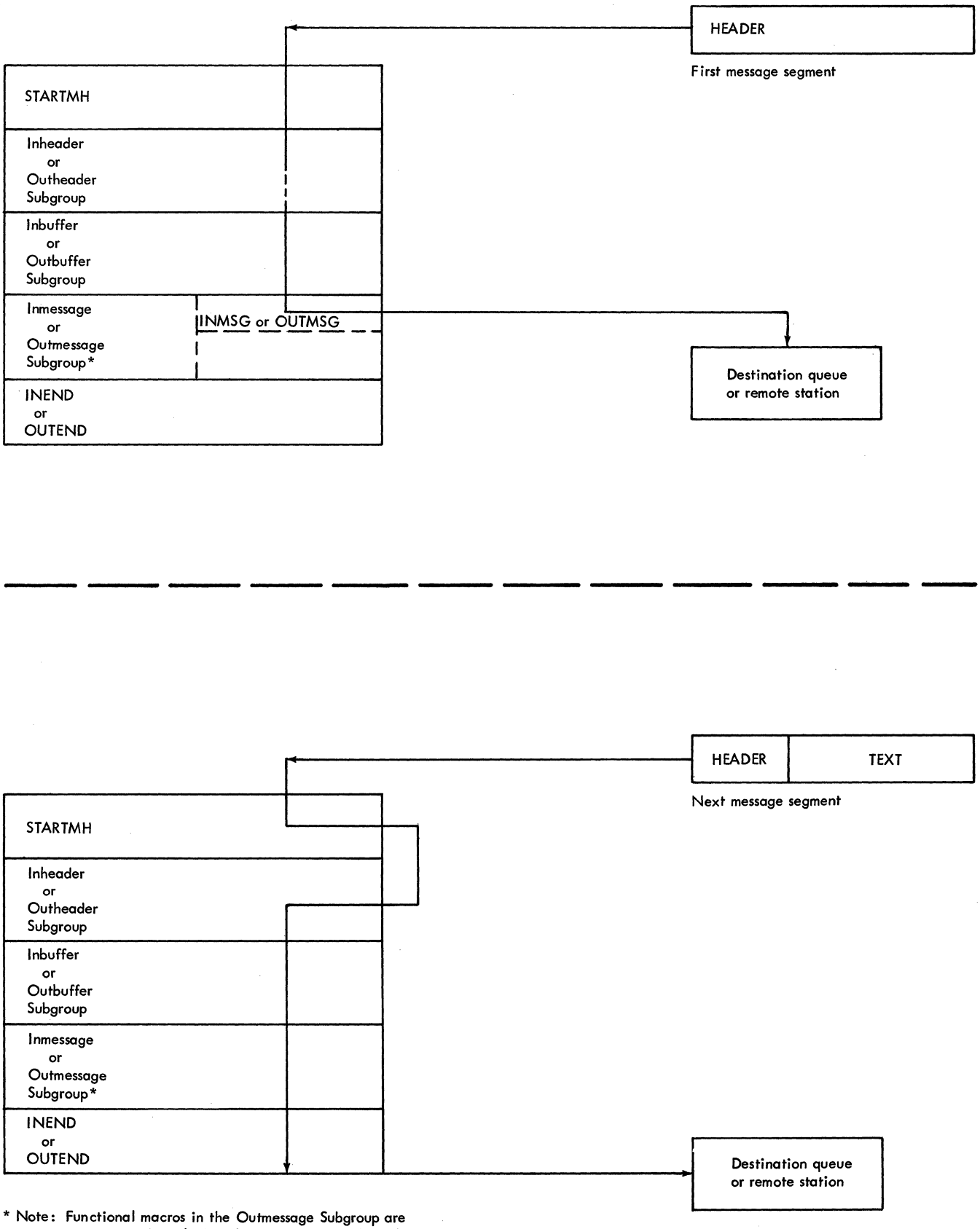
If the CODE, FORWARD, PRIORITY, or INITIATE macros are issued in an inheader subgroup handling multiple-buffer header segments, these macros must be specified early enough in the subgroup so that they act upon the first message segment. This also applies to PATH macros issued in an inheader or outheader subgroup, if all segments of the message are to be handled alike.





* Note: Functional macros in the Outmessage Subgroup are not executed until after the entire message has been sent.

Figure 15. Flow of a Two-segment Message with a Single-buffer Header through an MH



* Note: Functional macros in the Outmessage Subgroup are not executed until after the entire message has been sent.

Figure 16. Flow of a Two-segment Message with a Multiple-buffer Header through an MH

NOTE: Figure 16 contains only one of each kind of subgroup. For messages with multiple-buffer headers, the use of multiple inheader or outheader subgroups is severely restricted; all such subgroups must begin processing on the first message segment. In addition, if part of a multiple-buffer header is to be processed by an inheader subgroup and the rest is to be processed by an outheader subgroup, both subgroups must begin execution on the first message segment.

The execution of an inheader or outheader subgroup can begin only on the first segment of a message. This is because the INHDR or OUTHDR macro for a particular inheader or outheader subgroup causes all message segments except the first to bypass the subgroup. One inheader or outheader subgroup can handle a multiple-buffer header because the INHDR or OUTHDR macro does not get the opportunity to check segments other than the first (due to the way in which multiple-buffers are handled). If a second inheader or outheader subgroup is coded to begin execution midway through the second segment, it will never execute; its INHDR or OUTHDR macro will route incoming segments directly to the next delimiter.

NOTE: If an outbuffer subgroup precedes an outheader subgroup that processes more than one segment of a message having a multiple-buffer header, the outbuffer subgroup is executed for the first segment only.

Multiple-Buffer Header Processing Across Buffers

Macro	*N/A*	*Will Cross Buffers*	*Will Not Cross Buffers*	*Conditional (Note 1)*
CHECKPT	X			
CODE	(Note 2)			
COUNTER	X			
DATETIME	X			
FORWARD		(Note 3)	DEST in message	
INITIATE				X
LOCK				X
LOCOPT	X			
LOG	X			
MSGEDIT			X	
MSGFORM	X			
MSGLIMIT	X			
MSGTYPE				X
ORIGIN		(Note 4)		
PATH				X
PRIORITY		(Note 5)		
SCREEN				X
SEQUENCE	output only		input only	
SETEOF				X
SETSCAN		chars	integer POINT=BACK chars, RETURN=	
TERRSET	X			
UNLOCK				X

Note 1: Will cross if *conchars* is not specified, or if entire character string is in a subsequent buffer.
Note 2: Except that an operator command must be complete in a single buffer.
Note 3: Will cross if destination is in the macro or an option field and the macro is executed for the first buffer.
Note 4: Will cross but origin may not be known on dial lines for first buffer.
Note 5: Will cross if *conchars* not specified and priority level is in macro.

Variable Processing Within a Message Handler

The path of a message through a Message Handler may be varied dynamically using the PATH and MSGTYPE macro instructions. By permitting different operations upon different types of messages directed to the same Message Handler, these macros enhance the versatility of the Message Handler. By judiciously using PATH and MSGTYPE macros, the user can design a Message Handler that will handle messages having a variety of header formats, and that will perform different operations upon different types of messages, even when these different types are transmitted over the same line. Indeed, the user may in some cases be able to design a single Message Handler capable of processing all the messages that can be generated in a large TCAM-based telecommunications system performing a wide range of tasks.

The path of a message through a Message Handler may be varied in two ways. One of these involves the use of control characters in the message header, and the other involves the setting of switches, based on the control characters, that determine whether a given subgroup is to be executed for the message.

These switches, called *path switches*, are one-byte fields in the option table. The switches are initialized by an operand of the TERMINAL or TPROCESS macro and may subsequently be modified by a PATH macro or by a combination of OPTFIELD and DATOPFLD operator commands. An operand of a delimiter macro may specify that certain bits of a path switch are to be tested. If any of the specified bits are on, the subgroup introduced by the delimiter is executed; if none of the specified bits are on, control passes to the next subgroup. If a delimiter macro does not specify a path switch to be tested, its subgroup is executed unconditionally. Different delimiters may test different sets of path switches. For an example of the use of path switches and the PATH macro to control the routing of messages from subgroup to subgroup, see the discussion of the PATH macro.

By specifying, changing, and testing path switches, the user can determine which of the subgroups in an MH group are to be executed for a particular message. To control the path of a message *within* an inheader or outheader subgroup, the user may employ the MSGTYPE macro. The MSGTYPE macro compares a character or character string in the message header with a character or character string specified by a MSGTYPE operand. If the two characters or character strings match, the instructions between this MSGTYPE macro and the next MSGTYPE macro in the subgroup are executed (if this is the last MSGTYPE macro in the subgroup, all the remaining instructions in the subgroup are executed) and control is then passed to the next delimiter macro. If the two characters or character strings do not match, the instructions associated with this MSGTYPE macro are not executed, and control passes to the next MSGTYPE macro in this subgroup (or to the next delimiter, if this was the last MSGTYPE macro in the subgroup). A new comparison is made by each MSGTYPE macro to which control is passed. For an example of the use of the MSGTYPE macro to vary processing within a subgroup, see the description of the MSGTYPE macro.

The PATH macro controls the routing of a message *among* subgroups. The MSGTYPE macro controls the path of a message *within* an inheader or outheader subgroup.

Conditional Execution of Message Handler Functional Macros

Several MH functional macro instructions may request conditional execution dependent upon the existence of a control field in the message. These macros are INITIATE, LOCK, MSGTYPE, PATH, PRIORITY, SCREEN, SETEOF, and UNLOCK, with the optional operand *conchars*. *conchars* may consist of from one to eight nonblank characters and may be specified in unframed character format or with framing C' ' or CLn' ' characters, or in hexadecimal format with framing X' ' or XLn' ' characters.

These conditional characters specified in the macro are compared with the field in the message at the current location of the scan pointer. If the fields are identical, the macro will be executed and the scan pointer will be advanced to the last character of the field. If the characters do not match, the scan pointer is not moved and the macro is not executed.

If two or more macros in the same subgroup specify control character strings that are identical to a certain point but differ in length, and if there is any possibility that the same field in the message header will be checked for both strings, then these macros should be arranged according to decreasing length of their character strings. For example, if the user codes

```
INITIATE 1
LOCK 12
```

in his inheader subgroup, both macros will execute if the field in the message header contains 112. However, if the field contains 12, only the INITIATE macro will execute.

If the conditional characters are framed with CLn' ' or XLn' ' framing characters, *n* should agree with the actual count of characters. If *n* specifies a value greater than the actual count, it is possible that the macro may never be executed. For example, if a character string AB is defined as CL3'AB', the field is automatically padded to the right with a blank. If the BLANK= operand specifies BLANK=YES (or BLANK=X'40' or BLANK=C' '), a matching field can never be found. BLANK=YES states that blanks are not to be considered part of the character string when found in the header, but in this case the string used to determine execution contains a blank.

In the case of multiple buffer headers, the control characters must all be in the same buffer. The control characters may be entirely contained within the buffer in which the scan pointer is located when the comparison is begun, or they may be entirely contained within a subsequent buffer. They may not, however, be split between buffers.

If the sequence

```
MSGTYPE ABC
MSGTYPE AB
MSGTYPE A
```

is coded by the user, and if the characters being checked are AB and these are the last two bytes in the buffer, the first MSGTYPE executes just as if three characters were found but the compare was unequal. That is, the code following the first MSGTYPE is not executed and control passes to the second MSGTYPE macro. Execution of the second MSGTYPE finds two bytes, detects an equal compare, and passes control to the code following the second MSGTYPE. Note in this example, that even if a C is the next character beyond the AB in the message, the first MSGTYPE does not find the string because it is split between buffers.

If the string ABC is the next string in the message and is located entirely within the next buffer, execution of the first MSGTYPE detects that no characters remain in the current buffer. Processing of buffer fields in this buffer is deferred, (including the subsequent MSGTYPE processing). When the next buffer is passed to the Message Handler, execution of the first MSGTYPE resumes at the start of data in this next buffer and, because the string ABC is found, control passes to the code following the first MSGTYPE macro.

It is likely that the first result, where the string is split between buffers, is not the result desired by the user. To avoid such a result, either limit the header to a single buffer or avoid strings that are partially identical.

User Code in a Message Handler

The user may insert serially reusable assembler- or macro-language code in a Message Handler to supplement the facilities provided by TCAM. User-written code can be included as either an open or closed subroutine.

There are several reasons why the user might include such a subroutine. There may be no MH macro to process particular information he wishes included in his message headers. Or, he may wish to expand the scope of an MH macro (for example, to correct an invalid destination field detected by the FORWARD macro). A third case might be processing a header field in a manner entirely different from that in which the MH macro handles fields of this type. An example is inserting a date having a format different from the one used by the DATETIME macro.

General Requirements and Restrictions

The following requirements and restrictions apply to both open and closed user-written subroutines that supplement the functions provided by TCAM macros in a Message Handler:

1. All such subroutines must be serially reusable.
2. No executable code should be included within an inmessage or outmessage subgroup, or between such subgroups.
3. Branching from one Message Handler to another is not permitted.
4. System macros that issue an SVC should be avoided, unless the user is fully aware of the implications of using such macros in the TCAM system.
5. If the user provides a field or work area (as for the ERRORMSG, MSGGEN, and MSGEDIT macros), the field must be addressable by the MH. Such a field is addressable if placed after the OUTEND macro. If only one base register is used to establish addressability for the MH, the field must also be within 4096 bytes of the STARTMH macro in order to be addressable.
6. Nothing should be done that relinquishes control.
7. TCAM macros cannot be used in a closed subroutine.

Multiple-Buffer Header Considerations

When the MH is handling messages having multiple-buffer headers, user code within the inheader and outheader subgroups should test register 15 for a negative return code before executing any open user subroutines or branching to a closed user subroutine if the user subroutine to be executed depends upon certain data being in the buffer, or upon the location of the scan pointer. A negative return code indicates that the previous TCAM macro needed the next buffer but it was not available (for an understanding of how this situation could arise, see *Multiple-Buffer Header Handling* in this chapter). If a negative return code is detected, a branch should be made around a user subroutine that depends upon the presence of certain data in the header, or upon the scan pointer; such a subroutine is eventually executed on header fields in a subsequent message segment.

The USEREG= operand of the INTRO macro specifies the number of registers to be saved between header segments when user code is executed in an inheader or outheader subgroup that may handle multiple-buffer headers. The registers saved are sequentially ordered, beginning with general register 2. When the scan pointer comes to the end of a message segment and there is still code to be executed in the inheader or outheader subgroup processing the segment, TCAM saves the address of the next (unexecuted) inheader or outheader instruction and also saves the contents of the registers specified by USEREG=. The segment continues through the subgroup, but macros that depend upon the location of the scan pointer or upon specific data being present in header fields do not execute for the segment. When the second segment is ready for handling, the STARTMH macro routes it directly to the inheader or outheader instruction whose address was saved, and restores the contents of the saved registers. (See *Multiple-Buffer Header Handling* for more information on this topic.) Only the contents of those user registers specified by USEREG= are saved and restored.

Use of the USEREG= operand results in a larger MCP than would otherwise be the case. This operand should be coded only when an inheader or outheader subgroup that contains user code can expect to handle messages having multiple-buffer headers, with the user subroutine extending across buffers.

The user can determine the number of extra bytes of main storage that coding USEREG= will cost him by applying the following formula:

$$\Delta S=4R(L+T)$$

where

ΔS is the number of extra bytes added to the MCP.
R is the number of registers to be saved between buffers, as specified in the USEREG= operand of the INTRO macro.
L is the number of lines in the system on which are located stations whose TERMINAL macros omit the BFDELAY= operand.
T is the number of stations whose TERMINAL macros specify the BFDELAY= operand.

Including an Open Subroutine

A user-written open subroutine consisting of one or more assembler-language or system macro instructions may be included in-line in the inheader, inbuffer, outheader, and outbuffer subgroups of a Message Handler. TCAM macros may be included in an open subroutine. All registers except register 12 and 13 are immediately available for use in such a subroutine. If register 13 is accessed in the subroutine, its original contents must be saved and restored by the user. Register 12 should not be changed by user code, since it is the base register. If more than one base register is used, the other base registers must also be preserved.

When a user-written open subroutine is coded in an inheader or outheader subgroup that can handle messages having multiple-buffer headers, the contents of user registers will be lost if the header fields being processed by the user routine extend across more than one buffer. (To see why this is so, consider carefully *Multiple-Buffer Header Handling* in this chapter.) The user may specify that the contents of his registers be preserved in this case by suitably coding the USEREG= operand of the INTRO macro. When this operand is coded, the contents of the user's registers are saved when the scan pointer reaches the end of the first segment of a message having a multiple-buffer header, and restored to the user routine when the second segment arrives at the inheader or outheader subgroup.

Including a Closed Subroutine

A user-written closed subroutine may be included as a control section in the MCP; such a subroutine may be accessed by any Message Handler in the MCP, or as a result of an exit being taken that is specified by an INTRO, STARTMH, DCB, READY, ERRORMSG, or FORWARD macro. A closed subroutine cannot contain TCAM macros. When activating a closed subroutine, the user must provide his own linkages; he should save and restore the invoking Message Handler's registers. Figure 17 illustrates the flow of control between an MH and a user-written closed subroutine, and presents the recommended linkages.

Using LOCOPT to Locate An Option Field

The LOCOPT macro enables the user to obtain the address of any option field assigned to a particular station. The address of the desired field is placed in a user-specified register. A user-written routine may then examine and modify the contents of the option field.

Using SETSCAN to Locate a Header Field

The SETSCAN macro may be used to locate a portion of the message header for subsequent examination or processing by a user-written subroutine (but see Restriction No. 5 in the section *General Requirements and Restrictions*). For a detailed description of the capabilities of SETSCAN, see the discussion of the macro.

Two capabilities of SETSCAN are of particular interest with respect to user code:

- By coding MOVE=RETURN, the user may employ SETSCAN to locate a designated character string in the header and to place the absolute main-storage address of the last character of the string in a specified register, which may then be accessed by user code. When MOVE=RETURN is specified, the scan pointer is not actually moved, so the user need not worry about repositioning it. If this capability is to be utilized effectively, the character string to be examined must be located entirely within a single buffer unit, because buffer units are not usually contiguous in main storage. If a character string is split between two units, its two parts are likely to be at different locations in main storage. If the character string to be processed is divided between two buffer units, and the user knows where in the string the division occurs, he may treat the segments as separate character strings, issue a SETSCAN macro specifying MOVE=RETURN to find the address of each, and process each independently.
- SETSCAN may be used to determine the main-storage address of the header byte to which the scan pointer is currently pointing. This is done by specifying MOVE=RETURN and coding 0 as the *integer* operand. If the user codes

Operation	*Operand*
SETSCAN	0,MOVE=RETURN

the address of the current location of the scan pointer is returned in register 15.

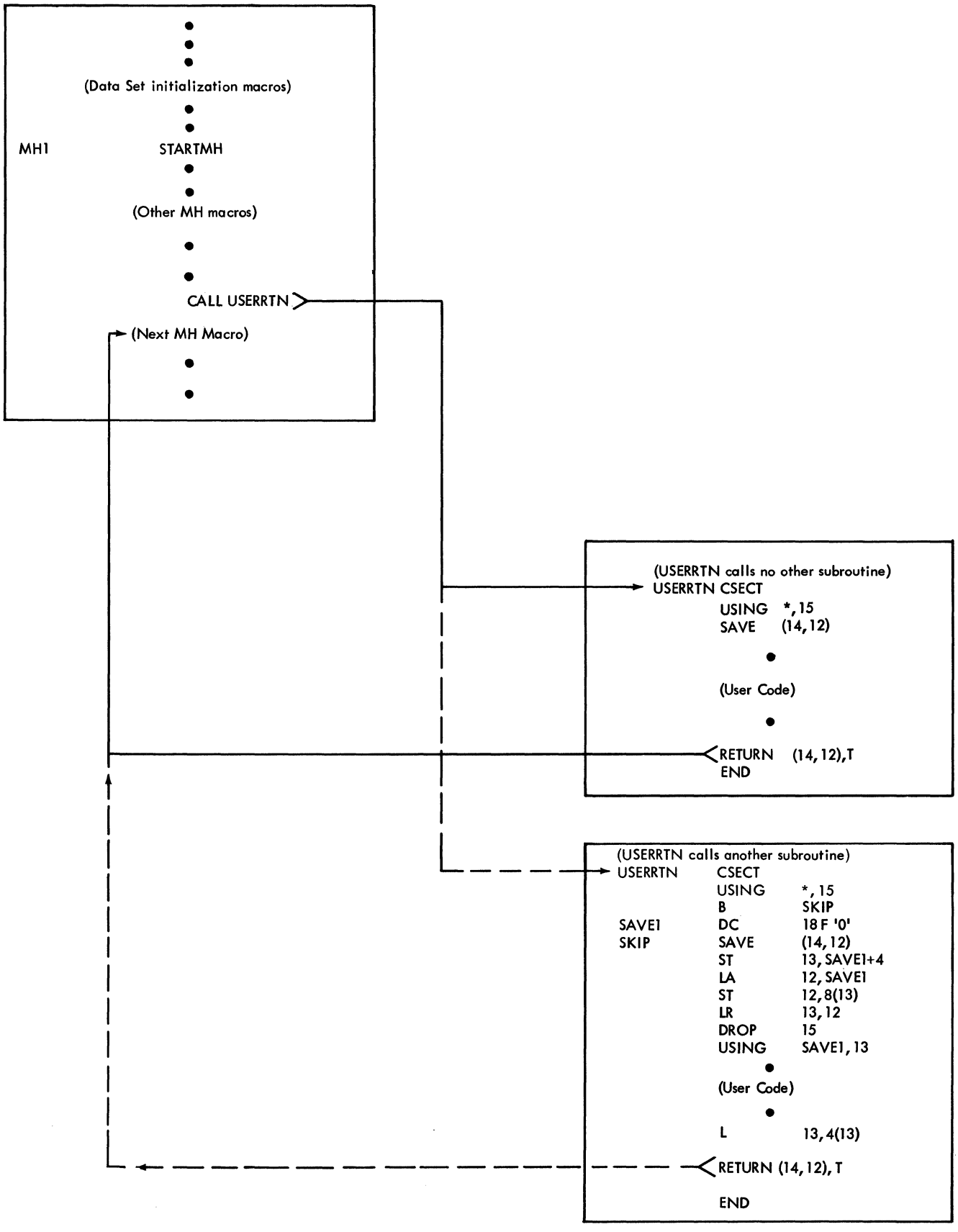


Figure 17. Activation of a Closed, User-written Subroutine

Using MSGTYPE to Locate a Header Field

User-written code may be included in inheader and outheader subgroups to interrogate and modify a field in a buffer of a message, and to interrogate but not modify a field that spans more than one buffer of a message header.

The next field in a buffer, the one immediately following the scan pointer, can be accessed from the buffer by use of a deliberately failing MSGTYPE macro as follows:

```
MSGTYPE C'XXXXXXXX'
MSGTYPE
```

Fields up to eight bytes in length can be accessed by this method. The number of characters in the MSGTYPE operand is the length of the field to be accessed. In this example, an eight-byte field is to be located. Care should be taken not to specify as the MSGTYPE operand any string that could be found in the header. If a matching string is found, the scan pointer is adjusted to point to its last byte.

If the string sought extends across buffers, it cannot be modified. This can be determined by examining the contents of the second byte of the AVT parameter area IEDPARM, addressable throughout the MCP. If this byte is less than the requested length, the field spans buffers. The byte at IEDPARM+1 is maintained in hexadecimal format.

If the field does not span buffers, it may be modified by the use of the MSGEDIT macro. Insertions before the string, removal of the string, or removal and replacement of the string may be performed by coding an appropriate MSGEDIT macro with the AT operand specified as SCAN (see the discussion of the MSGEDIT macro in this chapter). Insertion after the string may be performed by coding a SETSCAN macro with a count equal to the length of the string, followed by the desired MSGEDIT macro. It is recommended that a MSGEDIT macro specifying the string itself as the AT operand not be used, since all occurrences of the string found, not merely the current one, will cause the MSGEDIT function to be performed.

The following procedure allows the examination of a field that begins in one buffer and ends in a subsequent buffer (not necessarily the next buffer). It makes use of two parameters returned by the MSGTYPE function:

1. the characters found, whether matching or not, and whether as many as requested or not, are placed in the AVT work area IEDDOUBL;
2. the count of character found is placed in the second byte of the AVT parameter area IEDPARM.

The procedure may be varied depending on the length of the field to be examined. This example assumes the maximum, an eight-byte field.

```
SPLIT   MVI      COUNT,8          SET INITIAL LENGTH DESIRED
        LA       RWORK,WORK       POINT TO FIRST BYTE OF WORK
        CNOP     0,4              AREA
LOOP    MSGTYPE  C'XXXXXXXX'
        MSGTYPE
        CLI      IEDPARM+1,0      TEST IF ANY BYTES FOUND
        BE       NEXT             BRANCH IF NOT TO AWAIT NEXT
*                                 BFR
```

The operand of the MSGTYPE macro should specify as many characters as the string desired, and the operand should be such that a match cannot be found. Because no match is found, the MSGTYPE macro branches to the next delimiter. Therefore, a delimiter must be specified as a branch address. In this case, MSGTYPE is used.

```
         SR      REGA,REGA          CLEAR REGISTER FOR INSERT
         IC      REGA,COUNT         PICK UP DESIRED COUNT
         BCTR    REGA,0             DECREMENT FOR EXECUTE
         EX      REGA,MOVE          MOVE BYTES FOUND TO
*                                   WORKAREA
         LA      REGA,1(REGA)       RESTORE TRUE DESIRED COUNT
```

These instructions move the characters found to the work area. The MVC instruction (MOVE) is executed by the EX instruction so that the target address of the MVC can be modified.

```
         CLC     COUNT,IEDPARM+1    ARE ALL DESIRED BYTES FOUND
         BE      FOUND              BRANCH IF YES
*
```

The count field will be higher if not all the bytes are found.

```
         SR      REGB,REGB          CLEAR REGISTER FOR INSERT
         IC      REGB,IEDPARM+1     PICK UP NUMBER OF BYTES
*                                   FOUND
         AR      RWORK,REGB         ADJUST WORK AREA ADDRESS
```

The address register for the work area now points to the next byte to be filled, which will come from a subsequent buffer.

```
         SR      REGA,REGB          COMPUTE HOW MANY MORE
*                                   BYTES
         STC     REGA,COUNT         NEEDED AND RESET LENGTH
*                                   FIELD
         STC     REGA,SCAN+7        MODIFY SETSCAN PARAMETER
*                                   LIST
SCAN     SETSCAN 0
         B       LOOP               GO GET REMAINING BYTES
*
MOVE     MVC     0(0,RWORK),IEDDOUBL EXECUTED MOVE INSTRUCTION
WORK     DS      CL8
COUNT    EQU     LOOP+11
*
FOUND    DS      0H                 EXAMINE FIELD IN WORK
```

The SETSCAN function moves the scan pointer to point to the last byte in the buffer. The skip length is adjusted to the proper number by modifying the parameter list.

When the MSGTYPE function is executed again, the scan pointer is moved beyond the end of data in the buffer. The test following MSGTYPE must branch to the next sequential buffer processing required. Because of the setting of the scan pointer, that processing will not be performed on this buffer.

When the next buffer is passed to the Message Handler, execution of the MSGTYPE function will be started again. At this time, the remaining bytes desired will usually be accessed.

Note, however, that if the next buffer does not contain enough bytes to complete the count desired – if, for example, it contained only blanks – the buffer following it would be examined. That is, the field desired may actually be split over many buffers, and the procedure will still access it.

Using the PARM Parameter of the EXEC Job Control Statement

The user may wish to pass information to his user code by means of the PARM parameter of the EXEC job control statement (this capability is described in the OS publication *Job Control Language*). If the PARM= operand is specified, when control is passed to TCAM register 1 contains the address of a fullword, the low-order three bytes of which contain the address of a two-byte length field immediately followed by the data specified in the PARM parameter. When the INTRO macro is executed the address in register 1 is overlaid, but before this happens, INTRO stores the contents of register 1 in a fullword on a fullword boundary, from which it may be retrieved by user code. The name of the fullword is IEDSPLPT.

Message-Handler Macro Return Codes

During execution, certain MH macros cause a return code to be placed in a general register, usually register 15. The table below lists those TCAM macros whose return codes may be checked by user code in a Message Handler. The return code occupies the low-order byte in the register indicated; the rest of the register normally contains all zeros. Return codes of X'F6' are negative return codes; the high-order three bytes of the register contain binary ones. Some macros also return an address in a register; the locations and nature of such addresses are also indicated in the following table of MH macro return codes.

Macro	*Register*	*Return Code*	*Meaning*
COUNTER	15	X'00'	Good return
	15	X'FF'	Option field not found
DATETIME	15	X'00'	Good return
	15	X'04'	Insufficient reserve characters
FORWARD	15	X'00'	Good return
	15	X'04'	Invalid destination
LOCK	15	X'00'	Good return
	15	X'04'	Destination not specified
	15	X'08'	Destination not a process entry
LOCOPT			
a) if return requested in R15	15	Address of option field.	Good return
	15	X'00'	Option field not found
b) if return requested in user-specified register (USEREG)	15	X'00'	
	USEREG	Address of option field.	Good return
	15	X'04'	
	USEREG	Unchanged	Option field not found
LOG	15	X'00'	Good return
	15	X'04'	DCB or LOGTYPE entry named in macro not found
MSGEDIT	15	X'00'	Good return
	15	X'04'	No units available
MSGLIMIT	15	X'00'	Good return
	15	X'04'	Option field not found
ORIGIN	15	X'00'	Good return
	15	X'04'	Invalid origin
SCREEN	15	X'00'	Function not done
	15	Function byte	Good return

SEQUENCE			
a) macro issued in inheader subgroup	15	X'00'	Good return
	15	X'04'	Sequence number in message high
	15	X'08'	Sequence number in message low
	15	X'0C'	Originating station unknown
b) macro issued in outheader subgroup	15	X'00'	Good return
	15	X'04'	Insufficient reserve characters
SETSCAN			
a1) locate specified character string and return address in R15	15	Address of last character in string.	Good return
	15	X'00'	Specified character string not found in this buffer
	15	X'F6'	Scan pointer beyond end of buffer
a2) locate specified character string and return address in user-specified register (USEREG)	15 USEREG	X'00' Address of last character in string	Good return
	15 USEREG	X'04' Unchanged	Specified character string not found in this buffer
	15 USEREG	X'F6' Unchanged	Scan pointer beyond end of buffer
b1) skip *n* characters and return address in R15	15	Address of character skipped to	Good return
	15	X'00'	*n* greater than the number of characters remaining in this buffer
b2) skip *n* characters and return address in user-specified register (USEREG)	15 USEREG	X'00' Address of character skipped to	Good return
	15 USEREG	X'04' Unchanged	*n* greater than the number of characters remaining in this buffer
c) skip *n* characters backward	15	X'00'	Good return
	15	X'04'	*n* greater than the number of characters preceding the scan pointer in this buffer
d) Locate scan pointer address	15	Address of scan pointer	Good return
	15	X'F6'	Scan pointer beyond end of buffer

Message Translation

TCAM provides a facility for translating incoming messages from line code into EBCDIC and for translating outgoing messages from EBCDIC to line code. Translation is specified by issuing a CODE macro in the incoming and outgoing groups of an MH.

The user who does any appreciable amount of header analysis, or whose system includes stations using different line codes, will probably want to use TCAM's translation facility. Incoming translation must be specified for lines over which operator commands may be

entered, since the CODE macro is used by TCAM to check for such messages (see the discussion of the CODE macro for a way of selectively translating operator commands while leaving other messages entered on the same line untranslated).

Translation is not required in a message-switching application for which little or no header analysis is required, provided that the originating and destination stations are of the same type. The careful user may be able to avoid translating in other situations. The operands of most MH macros may be specified in hexadecimal format. By using the tables and information located in *Appendix G* the user may enter in his MH macro operands the hexadecimal representation of header fields that are in line code and thereby avoid having to translate. The user seeking to avoid translation should remember that the names entered in the terminal table (i.e., the names given to the TERMINAL, TLIST, TPROCESS, and LOGTYPE macros) must be specified in EBCDIC characters; no hexadecimal capability is provided for specifying these names.

The user may avoid translation of messages handled by a particular incoming or outgoing group of a Message Handler by omitting the CODE macro from that group. The user may avoid translation of messages received from or sent to the lines in a certain line group by coding a CODE macro having no operand and by specifying TRANS=EBCD in the line group DCB macro for the line group.

Translation is normally accomplished by means of tables provided by TCAM, although the user may provide his own translation tables if he wishes. The user providing his own tables should format the individual 256-byte tables as described in the example illustrating the use of the TRANSLATE instruction in the OS publication *Principles of Operation.* A user-defined translation table should consist of a fullword boundary, followed by a 256-byte table for translating from line code to EBCDIC, which is followed in turn by a 256-byte table for translating from EBCDIC to line code. The initial word should contain the address of the first byte of the second table.

Translation tables are provided for all stations supported by TCAM. The names of these tables are given in the following list. When one of these names, or the name of a user-specified table, is coded as part of the TRANS= operand of the line group DCB macro, incoming messages for this line group are translated from the specified line code to EBCDIC, and outgoing messages are translated from EBCDIC to the line code, when CODE macros are executed in the incoming and outgoing groups of the MH for the line group. The table specified by the DCB operand can be changed for messages to or from a particular line, station, or application program by entering a different table name in the *tablename* operand of the CODE macro and by using MSGTYPE macros or path switches to cause different CODE macros to be executed for different messages (see *Variable Processing within a Message Handler* in this chapter).

All of the characters in the character sets of each of the types of station supported by TCAM can be represented within the computer. However, some characters valid for one type of station may not be valid for another type, and some characters valid for a station may have no EBCDIC equivalents. The way in which TCAM handles these problems is described in the sections *Nonequivalent Characters* and *Substitutions* in *Appendix D. Internal and Transmission Code Charts.*

See *Appendix G. Device Dependent Considerations,* for specific information about the character sets for the:

- 1050 Data Communication System;
- 2260 Display System;
- 2740 Communications Terminal;
- TWX stations;
- WTTA terminals.

Names of Code Translation Tables Provided by TCAM

Table Name	*Type of Conversion*	*Terminal*
1030	1030 code to EBCDIC and back	IBM 1030
1050	1050 code to EBCDIC and back	IBM 1050
105F	1050 code to EBCDIC and back; converts incoming lowercase letters to uppercase	IBM 1050
1060	1060 code to EBCDIC and back	IBM 1060
2260	2260 code to EBCDIC and back	IBM 2260
2265	2265 code to EBCDIC and back	IBM 2265
2740	2740 code to EBCDIC and back	IBM 2740
274F	2740 code to EBCDIC and back; converts incoming lowercase letters to uppercase	IBM 2740
BC41	2741 BCD to EBCDIC and back	IBM 2741
EB41	2741 EBCDIC to EBCDIC and back	IBM 2741
CR41	2741 Correspondence Code to EBCDIC and back	IBM 2741
ITA2	5-level International Telegraph Alphabet No. 2 to EBCDIC and back	WTTA
ZSC3	5-level Figure Protected Code ZSC3 to EBCDIC and back	WTTA
TTYA	5-level (Baudot) code to EBCDIC and back	AT&T 83B3, WU 115A
TTYB	8-level TWX code to EBCDIC and back	AT&T 33/35 TWX (parity)
TTYC	8-level TWX code to EBCDIC and back	AT&T 33/35 TWX (nonparity)
6BIT	6-bit Transcode to EBCDIC and back	IBM 2780
ASCI	ASCII to EBCDIC and back	IBM 2770, 2780, S/360, Model 20
EBCD	No translation; coded for stations using EBCDIC transmission code, and when no translation is desired.	IBM 1130, 2770, 2780, S/360, Model 20

TCAM's Hold/Release Facility

TCAM provides the capability of temporarily suspending transmission of outgoing messages to a station. Transmission may be suspended either for a specified period of time or until the user chooses to resume outgoing traffic. Messages may be held either by the HOLD MH macro or by the operator command SUSPXMIT, and released by the RESMXMIT operator command, by the MRELEASE macro issued in an application program, or automatically at the expiration of the time interval specified by the HOLD macro.

HOLD is used to defer transmission of messages that should not be sent immediately because of error conditions at the destination station (the destination is the station for the message being processed by the outmessage subgroup when HOLD executes). If the macro is not used, messages that cannot be transmitted because the destination is temporarily out of order are treated as if they have been transmitted, even though they do not reach their destinations.

Once HOLD executes in an outmessage subgroup, no messages are sent to the destination station either until the interval specified as an operand of the macro expires, or until a RESMXMIT operator command or an MRELEASE application program macro is issued. Accumulated messages can be released by RESMXMIT or MRELEASE even though the specified time interval has not elapsed.

HOLD can be either unconditional or conditional based upon the setting of the message error record. HOLD until a release is issued can be used if a station unexpectedly fails. The error situation might be detected by a HOLD macro based on the message error record. The interval format can be used if a station in the system is scheduled for maintenance for a specific period of time. In this case, an unconditional HOLD with the INTVL= operand might be used.

If HOLD is issued in an incoming group, the station that will be held is the station that entered the message.

The HOLD macro cannot be executed for a station supported by main-storage only queues or for a station whose line is not open or has been opened idle. The operator command SUSPXMIT, which also causes an intercept, cannot be used unless a HOLD macro has been coded somewhere in the Message Handlers. If the operator control hold facility only is required, the HOLD macro coded in an MH can specify an impossible combination of errors in the mask associated with the message error record. This will ensure that the macro is never executed and will provide the operator control capability.

Design Steps

The design of a Message Handler and related control areas is a complex procedure that is difficult to codify according to a strict succession of steps. The difficulty arises from the profusion of functions available in TCAM and the variety of requirements of the installations. The following outline suggests a possible approach but does not pretend to be definitive.

1. Define the requirements of the application. Are messages to contain both header and text segments? Is an application program involved? How many characters will be in a normal message and in the longest message? Optimum buffer size depends on this.
2. Refer to the table of MH Functions and Macros Defining the Functions in a previous section of this chapter and the related text to determine which functional macro instructions might be used in the application. Study the detailed descriptions of these macros. Tentatively select those macros that provide the desired functions.
3. Design the message header, if applicable. Are the following fields necessary – origin, date, time, destination (single or multiple), priority? Should the field be entered by the terminal operator or inserted by the Message Handler? What should the program EOA characters be?
4. Start from a minimum Message Handler (required delimiter macros) and add the macros necessary to process the header.
5. Determine what validity checking is required and add the appropriate macro instructions.
6. Determine what error conditions need to be tested for and handled.
7. Determine what supplementary functions are desired – logging, initiate handling, message limit, etc.
8. Assemble the completed MCP and take a look at it before linkage editing.

No attempt should be made to write a Message Control Program in one step. A program should first be written, assembled, and tested that provides a very few of the desired functions. Other functions may be added as familiarity with the TCAM facilities grows and as the simpler programs are run successfully. For example, in a message switching application, a first program might include only the delimiter macros and the ORIGIN, SEQUENCE, CODE, and FORWARD functions. A second step might add the block checking function of the STARTMH macro, DATETIME, and some ERRORMSG functions. A third step might add multiple destinations, message counting and logging, and additional error handling. A final step could add the MSGTYPE or PATH functions to handle different message types.

The order in which macro instructions are specified requires thoughtful planning. It is important that some macro instructions be specified early enough in a subgroup so that they act on the first header buffer; these macro instructions are CODE, FORWARD, PRIORITY, and INITIATE (and PATH if all segments of the message are to be handled alike). In determining the relative placement of macros within the subgroup, the use of the scan pointer must be understood. (Note the sample Message Control Programs in the chapter *Putting the MCP Together.*)

Delimiter Macro Instructions

Delimiter macro instructions identify the beginning or the end of various groups and subgroups of a Message Handler. They also provide initialization (addressability) and control functions within an MH. The table below shows the various groups and subgroups and the delimiter macro-instructions that control their execution.

The STARTMH macro identifies the beginning of an MH and must be the first instruction in every MH. TCAM provides initialization by setting up base registers and addresses for an MH at this point. STARTMH code determines whether the message being processed is incoming or outgoing and directs the segment to the incoming or outgoing group accordingly. STARTMH handles end-of-block checking, if specified.

MH Groups, Subgroups, and Delimiter Macro Instructions

Message Handler		
Group	*Subgroup*	*Delimiter Macro*
		STARTMH
INCOMING GROUP	INHEADER SUBGROUP	INHDR
	INBUFFER SUBGROUP	INBUF
	INMESSAGE SUBGROUP	INMSG
		INEND
OUTGOING GROUP	OUTHEADER SUBGROUP	OUTHDR
	OUTBUFFER SUBGROUP	OUTBUF
	OUTMESSAGE SUBGROUP	OUTMSG
		OUTEND

STARTMH is the only delimiter macro that is always required. If the MH is to handle incoming messages, the INHDR, INEND, and OUTEND delimiter macros are required. If the MH is to handle outgoing messages, the OUTEND macro is required. Each of the remaining delimiters is required only if the user chooses to include in the MH the functional macros associated with that delimiter.

An inheader subgroup must be the first subgroup of the incoming group if an incoming group is present. An outheader subgroup may be specified before or after an outbuffer subgroup (if both are present). INEND and OUTEND identify the ends of the incoming and outgoing groups respectively.

If an MCP contains more than one MH, a LTORG instruction (described in the OS publication *Assembler Language*) should be coded immediately after the last delimiter macro (INEND or OUTEND) of each MH if in-line user code includes literals.

STARTMH Macro Instruction

The STARTMH macro:

- Establishes addressability for an MH,
- Directs messages to an incoming or outgoing group, as appropriate,
- Specifies whether line-control characters are to be left in messages,
- Checks for occurrence of hardware errors during message transmission,
- Handles user-detected logical errors,
- Specifies whether tete-a-tete interaction may occur between the computer and a station,
- Specifies whether end-of-block completion handling is to be used,
- Is required as the first macro in every MH.

STARTMH is required and must be the first macro instruction of every MH. If the STOP=, CONT=, CONV=, or LOGICAL= operand is specified, end-of-block (EOB) checking is performed. Basically, this checking consists of determining whenever an EOB, ETB, ETX, or EOT control character is received, whether certain types of transmission and user-specified logical errors have occurred; if so, the message is disposed of according to certain operands specified in STARTMH.

For an incoming message, EOB checking occurs before a buffer containing an EOB is processed by the MH. If a hardware error is detected and retry is possible, a retry operation is performed (see the *Glossary* for a definition of retry). No further MH handling occurs until the block is received again. If retry is not possible (because, for instance, the retry count is exhausted), either the error is ignored and the channel program is restarted to receive the next block, or transmission is terminated and the buffer continues through the MH, which processes it as the last buffer of the message.

After EOB checking (if any), and when it is full, a buffer containing an incoming message segment is passed to the appropriate subgroup. Depending upon how the user has coded the PCI= operand of the line group DCB macro, and upon whether or not his incoming message contains EOB control characters, the buffer may be deallocated and passed to the appropriate MH subgroup soon after it is filled, or it may not be passed to the appropriate subgroup until transmission has ceased on the line. (See *Dynamic and Static Buffer Allocation* in the chapter *Defining Buffers.*) A full buffer is deallocated whenever a program-controlled interruption occurs; if this is not provided for by the user, no deallocation occurs until an EOT control character is received.

For outgoing messages, EOB checking (if specified) is performed after each block is transmitted. No check is made for logical errors. The transmission of a particular block is deemed successful if the receiving terminal acknowledges that it has successfully received the block. Transmission errors detected by the terminal result in retries; once the retry count is exhausted, transmission is either terminated or allowed to continue, as for incoming messages. After transmission has terminated, control passes to the out-message subgroup, whose macros may then check the message error record for the message and take appropriate action.

See *Appendix G. Device Dependent Considerations*, for specific coding information concerning the 1030 Data Collection System, the 1060 Data Communication System, the 2770 Data Communication System, and the 2780 Data Transmission Terminal.

NOTE: If the user specifies dynamic buffer deallocation by the PCI= operand of the line group DCB macro, and if the block size for his incoming messages is greater than his buffer size for incoming messages, segments containing transmission errors may be processed by the inheader and inbuffer subgroups of the MH before the EOB-checking routine detects the errors. In this case, when the EOB-checking routine detects the errors, segments in this block that have been enqueued are dequeued or ignored, and the input sequence number is decremented if it was incremented by a canceled segment. Dequeuing and sequence-number adjustment are done automatically by TCAM. However, any option fields that were updated on the basis of data in the canceled segments remain updated, and if the canceled segments were logged they remain on the logging medium.

NOTE: When the INITIATE macro is executed in the inheader subgroup handling an incoming message, no EOB checking is performed for that message, either on the incoming or the outgoing side, regardless of what is specified by STARTMH operands.

Name	*Operation*	*Operand*
mhname	STARTMH	LC={IN / OUT} [,{STOP / CONT}={YES / (opfield,switch)}] [,CONV={YES / (opfield, switch) / NO}] [,LOGICAL={(opfield) / (opfield 1, switch, opfield 2)}] [,BREG={integer / 1}]

mhname

Function: Name of the macro and of the Message Handler.
Default: None. This name must be specified.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).
Notes: Must be the same as *mhname* specified in the MH= operand of the DCB for the line group that uses this Message Handler.

LC={IN / OUT}

Function: Specifies whether line-control characters in a start-stop message or a BSC message in nontransparent mode are to be removed.
Default: None. This operand must be specified.
Format: IN or OUT
Notes: OUT causes TCAM to remove EOA and EOB line-control characters from incoming messages entered at a start-stop station, and to remove STX, ETX and ETB line control characters from incoming messages entered at a BSC station. EOT line control characters are not removed when OUT is specified. EOB and ETB line control characters are not removed when CONV=YES is specified, regardless of how LC= is coded. Line control characters are not removed until after the message segment is in a buffer; therefore, the buffer must be large enough to accomodate line control.

The ITB control character is not considered by TCAM to be a line-control character, and is not removed when OUT is specified.

IN causes the line control to remain in incoming messages (unless such messages are in transparent mode, in which case "real" line control characters are removed regardless of how this operand is coded).

STOP={YES / (opfield,switch)}

Function: When a message block is found to be in error, this operand (conditionally) specifies that once the retry count is exhausted, transmission of this message is to be terminated. The error may be a hardware error or may be a user-detected logical error if the LOGICAL= operand is also specified.

Default: None. Specification optional.
Format: YES or (opfield,switch). *opfield* must conform to the rules for assembler language symbols, and must be the name of a one-byte option field defined by an OPTION macro. *switch* may be either decimal or hexadecimal. If hexadecimal format is used, framing X'' characters must be specified.
Maximum: 255 or a one-byte hexadecimal field for *switch*.
Notes: YES specifies that transmission is to be terminated unconditionally. *(opfield, switch)* specifies that transmission is to be terminated if any of the bits on in the switch are also on in the option field.

When transmission is terminated because of an error detected by EOB checking, that portion of the message that has been received (or sent) continues through the incoming (or outgoing) group of the MH, where it is treated as if it were a complete message. The user may issue certain error-handling macros in the inmessage (or outmessage) subgroup of the MH that test bit 25 of the message error record and dispose of the message according to his specifications if bit 25 (which indicates that an error occurred during transmission of data) is on. If this operand is omitted, end-of-block checking is not done.

CONT= { **YES** / **(opfield,switch)** }

Function: When a message block is found to be in error, this operand (conditionally) specifies that once the retry count is exhausted, transmission of this message is to be continued.
Default: None. Specification optional.
Format: YES or (opfield,switch). *opfield* must conform to the rules for assembler language symbols, and must be the name of a one-byte option field defined by an OPTION macro. *switch* may be either decimal or hexadecimal. If hexadecimal format is used, framing X'' characters must be specified
Maximum: 255 or a one-byte hexadecimal field for *switch*.
Notes: YES specifies that transmission is to continue unconditionally. *(opfield,switch)* specifies that transmission is to continue if any of the bits on in the switch are also on in the option field.

When an error occurs, a bit in the message error record is set on. Message segments are sent to the appropriate MH group just as if no EOB error had been found. If this operand is omitted, end-of-block checking is not done.

CONV= { **YES** / **(opfield,switch)** / **NO** }

Function: Specifies whether EOB completion handling is to be used for the station, and (in conjunction with a LOCK macro) whether tete-a-tete interaction is in effect for the station.
Default: CONV=NO
Format: YES or NO or (opfield,switch). *opfield* must conform to the rules for assembler language symbols, and must be the name of a one-byte option field defined by an OPTION macro. *switch* may be either decimal or hexadecimal. If hexadecimal format is used, framing X'' characters must be specified.
Maximum: 255 or a one-byte hexadecimal field for *switch*.
Notes: YES specifies that tete-a-tete interaction is to be used unconditionally, and that a logical block of data being entered by a station is to be treated by TCAM as if it were a complete message.

NO specifies that tete-a-tete interaction and EOB-completion handling are not to be used. *(opfield,switch)* specifies that tete-a-tete interaction and EOB-completion handling are to be used if any of the bits on in the switch are also on in the option field. If this operand is specified, STOP= or CONT= must also be specified. For an explanation of tete-a-tete interaction, see *TCAM's Inquiry/Rapid Response Facility* in *Writing TCAM-Compatible Application Programs.*

LOGICAL= { **(opfield)** / **(opfield1,switch,opfield2)** }

Function: Specifies whether a user-written routine is to be given control (conditionally) to test for logical errors in incoming messages on a block-by-block basis. Such errors might include a formatting error in a card or an inquiry addressed to the wrong application program.
Default: None. Specification optional.

Format: (opfield) or (opfield 1,switch,opfield 2). *opfield* and *opfield2* must conform to the rules for assembler language symbols, and must be the names of option fields defined by OPTION macros. *switch* may be either decimal or hexadecimal. If hexadecimal format is used, framing X'' characters must be specified.
Maximum: 255 or a one-byte hexadecimal field for *switch.*
Notes: (*opfield*) specifies that a user-written routine is to be given control unconditionally. *opfield* refers to a four-byte option field, the high-order byte of which indicates that an error has been found. The low-order three bytes are the address of the routine to be given control. (*opfield1,switch,opfield2*) specifies that the user-written routine in *opfield1* is to be given control conditionally if any of the bits on in the switch are also on in the one-byte *opfield2.*

If this operand is specified, STOP= or CONT= must also be specified. The user may initialize the routine-name option field by coding a V-type address constant naming his routine as part of the OPDATA= operand of the TERMINAL or TPROCESS macro. Upon return from the user routine, STARTMH examines the high-order byte of the field. If the byte is not zero and if the STOP= operand specifies that transmission is to be terminated, transmission is terminated. If the byte is zero, or if the CONT= operand is in effect, operations are restarted on the line.

The user routine must save and restore registers 2 through 12, and must not alter the contents of register 13 and 14. The information contained in the following registers upon entry to the user routine may be of interest to the user:

Register 1 contains the address of the four-byte option field containing the one-byte error indicator followed by the address of the user routine.

Register 4 contains the address of the LCB (line control block), an internal TCAM control area described in the TCAM PLM.

Register 6 contains the address of the last buffer in the block of the data (the buffer containing the EOB); this is the only buffer in this block that may be tested for logical errors by the user routine.

Register 8 contains the address of the SCB (station control block), an internal TCAM control area described in the *TCAM PLM.*

Register 13 contains the address of a TCAM save area, and must not be altered.

Register 14 contains the return address for the calling routine and must not be altered.

Register 15 contains the address of the entry point for the user routine.

BREG={**integer** / **1**}

Function: Specifies the number of base registers desired for this MH.
Default: BREG=1
Format: An unframed decimal integer between 1 and 11.
Maximum: 11
Notes: One base register is required for each 4096 bytes in the MH. Assignment begins with register 12 and works downward. If BREG=3 is coded, for instance, registers 12, 11 and 10 are assigned as the base registers for the first three 4096-byte blocks of this Message Handler.

INHDR Macro Instruction

The INHDR macro:

- Identifies the beginning of an inheader subgroup,
- Tests a path switch to allow alternative courses of action,
- Is required as the first macro of any incoming group.

INHDR identifies the beginning of an inheader subgroup, the functional macros in which are concerned only with incoming header segments. An inheader subgroup must be the first subgroup in the incoming group. Text segments are passed to the first inbuffer subgroup.

An incoming message segment is tested by INHDR to determine whether it is a header or text segment (the first segment of any message is always considered to be a header segment). If it is a text segment or a canceled message, the segment is passed to the next subgroup; if it is a header segment, the inheader subgroup is executed.

If the PATH= operand of INHDR is coded, INHDR examines a one- byte path switch in a field of the option table. If any of the bits specified by INHDR are on in the path switch, this subgroup is executed. If none of the bits are on, control is directed to the next subgroup. If INHDR does not specify an operand, this subgroup is executed unconditionally. For a more complete description of the path switch and its function, see *Variable Processing within a Message Handler* in this chapter.

Name	*Operation*	*Operand*
[symbol]	INHDR	[PATH=(name,switch)]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

PATH= (opfield,switch)

Function: Specifies conditional execution of this macro.
Default: None. Specification optional.
Format: (opfield,switch). *opfield* must conform to the rules for assembler language symbols, and must be the name of a one-byte option field defined by an OPTION macro. *switch* may be either decimal or hexadecimal. If hexadecimal format is used, framing X'' characters must be specified.
Maximum: 255 or a one-byte hexadecimal field for *switch.*
Notes: If this operand is not specified, the subgroup is executed unconditionally.

INBUF Macro Instruction

The INBUF Macro:

- Identifies a subgroup that handles incoming message buffers,
- Tests a path switch to allow alternative courses of action,
- Is optional in the incoming group.

INBUF identifies the beginning of an inbuffer subgroup, which contains instructions concerned with both header and text portions of incoming messages.

If the PATH= operand of INBUF is coded, INBUF examines a path switch in a field of the option table. If any of the bits specified by INBUF are on in the path switch, this subgroup is executed. If none of the bits specified by INBUF are on in the path switch, processing goes to the next subgroup. If INBUF does not specify an operand, this subgroup is executed unconditionally.

Name	*Operation*	*Operand*
[symbol]	INBUF	[PATH= (opfield,switch)]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

PATH= (opfield,switch)

Function: Specifies conditional execution of this macro and its subgroup.
Notes: For details concerning this operand, see the description of the INHDR macro.

INMSG Macro Instruction

The INMSG macro:

- Identifies the beginning of an inmessage subgroup,
- Tests a path switch to allow alternative courses of action,
- Is required as the first macro in an inmessage subgroup,
- Is optional in the incoming group.

INMSG identifies the beginning of an inmessage subgroup. The functional macros associated with this subgroup are executed after an entire message or block has entered the system. Inmessage subgroups are specified after other subgroups in the incoming group. No user-written code should be included in an inmessage subgroup, or between such subgroups.

If the PATH= operand of INMSG is coded, INMSG examines a path switch in a field of the option table. If any of the bits specified by INMSG are on in the path switch, this subgroup is executed. If none of the bits specified by INMSG are on, processing branches to the next subgroup. If INMSG does not specify an operand, this subgroup is executed unconditionally. Only one inmessage subgroup per message can be executed.

INMSG causes empty buffer units at the end of a buffer handled by this Message Handler to be deallocated before the contents of the buffer are queued for a destination. Deallocated units are returned to the available unit queue. When the inmessage subgroup is not included in a Message Handler, this deallocation function is performed by the INEND macro.

Name	*Operation*	*Operand*
[symbol]	INMSG	[PATH= (opfield,switch)]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

PATH= (opfield,switch)

Function: Specifies conditional execution of this macro and its subgroup.
Notes: For details concerning this operand, see the description of the INHDR macro.

INEND Macro Instruction

The INEND macro:

- Identifies the end of the incoming group of an MH,
- Is required as the last macro of any incoming group.

INEND identifies the end of the instruction sequence that processes incoming messages. One and only one INEND macro is required for each MH with an incoming group ; it must be the last macro in the incoming group. No operand is required.

Name	*Operation*	*Operand*
[symbol]	INEND	(no operands)

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

There are no operands for this macro.

OUTHDR Macro Instruction

The OUTHDR macro:

- Identifies the beginning of an outheader subgroup,
- Tests a path switch to allow alternative courses of action.
- Is optional in an outgoing group.

OUTHDR identifies the beginning of an outheader subgroup, which is concerned only with the header portions of outgoing messages and, if included, may be either before or after an outbuffer subgroup in the outgoing group.

An outgoing segment is tested to see whether it is a header or a text segment. The outheader subgroup is executed only on a header segment; it is bypassed if the segment contains text only.

If the PATH= operand of OUTHDR is coded, OUTHDR examines a path switch in a field of the option table. If any of the bits specified by OUTHDR are on in the path switch, this subgroup is executed. If none of the bits are on, control passes to the next subgroup. If OUTHDR does not specify an operand, this subgroup is executed unconditionally.

Name	*Operation*	*Operand*
[symbol]	OUTHDR	[PATH= (opfield,switch)]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

PATH= (opfield,switch)

Function: Specifies conditional execution of this macro and its subgroup.
Notes: For details concerning this operand, see the description of the INHDR macro.

OUTBUF Macro Instruction

The OUTBUF macro:

- Identifies a subgroup that handles outgoing message buffers,
- Tests a switch to allow alternative courses of action,
- Is optional in an outgoing group.

OUTBUF identifies the beginning of an outbuffer subgroup that contains instructions concerned with both header and text portions of outgoing messages. If included, an outbuffer subgroup may be located either before or after an outheader subgroup in the outgoing group.

If the PATH= operand of OUTBUF is coded, OUTBUF examines a path switch in a field of the option table. If any of the bits specified by OUTBUF are on in the path switch, this subgroup is executed. If none of the bits specified by OUTBUF are on, control passes to the next subgroup. if OUTBUF does not specify an operand, this subgroup is executed unconditionally.

Name	*Operation*	*Operand*
[symbol]	OUTBUF	[PATH= (opfield,switch)]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

PATH= (opfield,switch)

Function: Specifies conditional execution of this macro and its subgroup.
Notes: For details concerning this operand, see the description of the INHDR macro.

OUTMSG Macro Instruction

The OUTMSG macro:

- Identifies the beginning of an outmessage subgroup of an MH,
- Tests a path switch to allow alternative courses of action,
- Is required as the first macro in an outmessage subgroup,
- Is optional in an outgoing group.

OUTMSG identifies the beginning of an outmessage subgroup, which is executed only after an entire block or message has been sent. Outmessage subgroups are specified after other subgroups in the outgoing group.

If the PATH= operand of OUTMSG is coded, OUTMSG examines a path switch in a field of the option table. If any of the bits specified by OUTMSG are on in the path switch, this subgroup is executed. If none of the bits specified by OUTMSG are on, control passes to the next subgroup. If OUTMSG does not specify an operand, this subgroup is executed unconditionally. Only one OUTMSG subgroup per message can be executed.

OUTMSG causes empty units at the end of buffers handled by this outgoing group to be deallocated and returned to the available unit queue. If an outmessage subgroup is not coded, this deallocation function is performed by OUTEND.

Name	*Operation*	*Operand*
[symbol]	OUTMSG	[PATH= (opfield,switch)]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

PATH= (opfield,switch)

Function: Specifies conditional execution of this macro and its subgroup.
Notes: For details concerning this operand, see the description of the INHDR macro.

OUTEND Macro Instruction

The OUTEND macro:

- Identifies the end of any outgoing group,
- Is required as the last macro in any outgoing group.

OUTEND identifies the end of the instruction sequence that processes outgoing messages. One OUTEND is required for each outgoing group; it must be the last macro in the group. No operands are required.

Name	*Operation*	*Operand*
[symbol]	OUTEND	(no operands)

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

There are no operands for this macro.

Functional Macro Instructions

This section describes the functions provided by the MH macro instructions. The discussion of each macro begins with a capsule summary of its functions. The functions of the macro are then described in detail, with a discussion of related topics necessary to an understanding of these functions.

The coding of the macro is then described, using a boxed illustration. The formats of the macro illustrations and the symbols used are shown in *Appendix A*. General rules for interpretation of the operand descriptions are also provided in *Appendix A*, to which the reader should refer.

CANCELMG Macro Instruction

The CANCELMG macro:

- Cancels messages either unconditionally or when certain errors occur,
- Is optional in an inmessage subgroup (and is permitted in no other subgroup),
- If specified, must be the first functional macro executed in the subgroup (one or more CANCELMG macros may be specified in the same subgroup).

CANCELMG causes immediate cancellation of a message if any errors specified by the error mask operand are also set in the message error record (see *Appendix B* for a description of the message error record). A canceled message is not sent to its destination. If there are multiple destinations, the canceled message is not sent to any of them. The ERRORMSG or MSGGEN macro may be used to notify the terminal operator of the error, or the REDIRECT macro may be used to send the message that is in error to a selected destination.

If an all-zero mask is specified, or if the mask is omitted, the message is canceled unconditionally.

NOTE: CANCELMG should not be executed for a message if an INITIATE macro has been executed for that message.

If the CANCELMG macro is executed in the inmessage subgroup for a lock mode message, the lock is not broken and the terminal will be repolled.

Name	*Operation*	*Operands*
[symbol]	CANCELMG	[mask] [,CONNECT={AND / OR}]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*.

mask

Function: Specifies the five-byte bit configuration used to test the message error record for the message (the message error record is described in *Appendix B*).
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing characters must be specified. If X' ' is used, leading *zeros* must be coded. If XL5' ' is used, leading zeros may be omitted.
Maximum: 16777215 or a hexadecimal field five bytes in length.
Notes: Omitting the operand or specifying an all-zero mask causes unconditional execution.

CONNECT={AND / OR}

Function: Specifies the type of logical connection to be made between the mask and the message error record.
Default: CONNECT=OR
Format: AND or OR
Notes: AND specifies that the macro is to be executed only if *all* of the bits specified by *mask* are on in the message error record.

OR specifies that the macro is to be executed if *any* bit specified by *mask* is on in the message error record.

Example 1:
```
CANCELMG X'0000080100',CONNECT=AND
```
specifies that the message is to be canceled only if bits 20 and 31 of the message error record are *both* on.

Example 2:
```
CANCELMG 524544,CONNECT=OR
```
specifies that the message is to be canceled if either bit 20 or bit 31 of the message error record is on.

CHECKPT Macro Instruction

The CHECKPT macro:

- Causes an incident checkpoint record to be taken of the option fields for the originating or destination station or application program,
- May be coded in any subgroup of the Message Handler.

When coded in an inheader, inbuffer, or inmessage subgroup, the CHECKPT macro causes an incident checkpoint record to be made of the option fields assigned to the originating station or application program. This checkpoint record is taken after the entire incoming group has executed and the message has been enqueued, so that the option fields reflect the fact that a message has been processed by the incoming group.

When coded in an outheader, outbuffer, or outmessage subgroup, CHECKPT causes an incident checkpoint record to be taken of the option fields assigned to the destination station or application program. This checkpoint record is taken after the entire outgoing group has been executed and the message has been sent; the option fields reflect the fact that a message has been sent by the outgoing group.

If a message segment goes through any subgroup in which a CHECKPT macro is executed, an incident checkpoint record is made after that message has been completely handled by the appropriate MH group. Only one record per message is made, even if more than one CHECKPT macro is coded in the group. If no CHECKPT record is coded in a group, no incident checkpointing record is made when the message leaves the group.

For more information on TCAM's checkpoint facility, see the chapter *Using TCAM Service Facilities.*

The CHECKPT macro has the following format:

Name	*Operation*	*Operand*
[symbol]	CHECKPT	(no operands)

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

The CHECKPT macro has no operands.

CODE Macro Instruction

The CODE macro:

- Translates the data in the buffer being handled,
- Tests for operator commands,
- Is optional in the inheader, inbuffer, outheader and outbuffer subgroups (and not permitted in any other subgroup),
- May be issued at any point in the subgroup.

CODE causes the message segment being handled to be translated. If specified in an inheader or inbuffer subgroup, translation is from line code to EBCDIC; if specified in an outheader or outbuffer subgroup, translation is from EBCDIC to the line code. The line code to be used is specified by the TRANS= operand of the DCB macro or by an operand of the CODE macro (which overrides the table specified in the DCB macro).

If CODE is included in a subgroup, and any segments of a message are processed by that subgroup, the *entire message* is translated. Macros issued prior to CODE in the incoming group act on message characters that are in line code, while macros issued subsequent to CODE act on message characters that are in EBCDIC. The converse is true for the outgoing group. If CODE is not included in the incoming group, incoming messages are not translated; if CODE is not included in the outgoing group, outgoing messages are not translated.

NOTE: Once a message has been translated by a CODE macro executed in a subgroup of an incoming or outgoing group, care should be taken that no segment of it is routed through another subgroup when the second subgroup also contains a CODE macro. The second CODE macro would "translate" the message into gibberish.

The CODE macro permits flexibility of handling of buffers with respect to translation by overriding the translation table specified for the line group.

CODE tests for operator commands, and transfers control accordingly. If operator commands may be entered by any station on a line, then a CODE macro should be issued in the inheader subgroup of the MH handling incoming messages on that line. If the LC= operand of the STARTMH macro is coded LC=OUT (i.e., if line-control characters are to be automatically stripped from incoming messages), then CODE should be the first functional macro issued in the inheader subgroup for a line on which operator commands may be entered. If STARTMH is coded LC=IN (i.e., if line-control characters are not to be removed from incoming messages by TCAM), then a SETSCAN macro should be issued immediately prior to CODE. The SETSCAN macro should move the scan pointer to the last initial line-control character (except the machine EOA, which is automatically skipped by STARTMH). The scan pointer should be positioned on the last such character.

NOTE: The user at a station may wish to enter one or more characters in front of the character string identifying his operator command as such when he enters it at a station. This is permissible as long as the user sets his scan pointer to the nonblank character immediately preceding the operator control character string before issuing CODE.

The CODE macro must be issued in the inheader subgroup handling messages from a station if operator commands may be entered by that station. However, the user may not wish to translate ordinary messages entered at the station. One way to avoid having to translate every message is as follows (assume that line code is removed from incoming messages):

Code a special inheader subgroup as the first subgroup of the incoming group; this special subgroup may consist of a MSGTYPE macro followed by a CODE macro. Have the MSGTYPE macro look at the first field in each incoming message in line code, and execute only if this field consists of some specific character –for instance, A. Enter A before the identification sequence of each operator command. If the first character of a message is A, the CODE macro will execute and the message will be translated – otherwise, control will be passed to the next delimiter, which may be another inheader subgroup designed to handle ordinary messages in line code.

Example:
The following code might be used to check for operator commands entered at a 1050 station, and cause each incoming message to be translated only if it is an operator command. It is assumed that line code is removed from incoming messages and that the operator at the station enters an A immediately in front of the identification sequence for an operator command.

Name	*Operation*	*Operands*
OCMH	STARTMH	LC=OUT,STOP=YES
OCCHK	INHDR	
	MSGTYPE	X'E2'
	CODE	
NONOC	INHDR	
	INBUF	
	INMSG	
	INEND	

An incoming message enters the first inheader subgroup. If A (which is X'E2' in 1050 line code) is the first character in the message, CODE is executed. If this is an operator command, the CODE macro causes it to be handled as such, and it never reaches the second inheader subgroup. A message that does not begin with A is not translated and is passed to the second inheader subgroup, which contains macros that handle ordinary (non-operator-control) messages.

The CODE macro has the following format:

Name	*Operation*	*Operand*
[symbol]	CODE	[{tablename / NONE / register}]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

{tablename / NONE / (register)}

Function: Specifies the type of translation to be done.
Default: None. Specification optional.
Format: tablename, (register) or NONE. *tablename* must either be one of the names permitted for the TRANS= operand of the DCB macro or the name of a user-defined table that conforms to the rules for assembler language symbols. *(register)* must specify a decimal integer between 2 and 11.
Notes: If this operand is omitted, the table used for translation is that specified by the TRANS= operand of the DCB macro.

If NONE is specified, the message is not translated. NONE can be used to check for operator commands when the station transmits in EBCDIC.

If *(register)* is specified, the register must previously have been loaded with the address of the table to be used.

A user-defined translation table must consist of a fullword on a fullword boundary, followed by a 256-byte table for translating from line code into EBCDIC, followed by a 256-byte table for translating from EBCDIC into line code. The first word must contain the address of the first byte of the second table. The high-order byte of the first word must be zero.

COUNTER Macro Instruction

The COUNTER macro:

- Maintains a count of complete messages or of message segments received from or sent to a station,
- Is optional in inheader, inbuffer, outheader, and outbuffer subgroups.

COUNTER enables the user to maintain four types of count. The position of the COUNTER macro within an MH determines which type of count is maintained. COUNTER may appear:

- In the inheader subgroup to count incoming messages for each originating station;
- In the inbuffer subgroup to count incoming message segments for each originating station;
- In the outheader subgroup to count outgoing messages for each destination station;
- In the outbuffer subgroup to count outgoing message segments for each destination station.

Any one or more of the counts may be maintained by including COUNTER in the appropriate subgroups; within each subgroup, it may appear at any point.

For each COUNTER macro, the user must define, by an OPTION macro, a halfword option field for the appropriate station or component. This provides space for maintaining the counters.

The use of COUNTER is optional. If it is used in an inheader or inbuffer subgroup and the stations for which it maintains counts are on a switched line and do not have unique ID sequences, and if a calling station entering a message is not identified by an ORIGIN macro before COUNTER is executed, the option field associated with the related line entry in the terminal table will be referred to.

NOTE: The COUNT may not be exact since recalled messages (from ERRORMSG, for instance) and messages from buffered stations will be counted twice.

Name	*Operation*	*Operand*
[symbol]	COUNTER	opfield

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

opfield

Function: Specifies the name of the halfword option field for the station or component in which the count is to be maintained.
Default: None. This operand must be specified.
Format: Must conform to the rules for assembler language symbols and must be the name of a halfword option field defined by an OPTION macro.
Notes: The field contains a binary count up to a maximum of 65535. When the maximum count has been reached, the count is reset to zero for the next message or segment counted.

The user may gain access to the field at any time to determine or reset the count (by operator commands or by user code including the LOCOPT macro.) The count is initially set using the OPDATA= operand of the TERMINAL or TPROCESS macro.

If the option field is not found, COUNTER does not execute and control passes to the next MH instruction. A return code of X'FF' in the low-order byte of register 15 indicates that COUNTER did not execute.

CUTOFF Macro Instruction

The CUTOFF macro:

- Specifies the maximum allowable length of incoming messages,
- Checks for incoming buffers filled with identical characters,
- Is optional in the inbuffer subgroup (and not permitted in any other subgroup),
- Is normally issued prior to a related ERRORMSG macro.

CUTOFF specifies the maximum number of characters allowed in an incoming message. If the maximum number is reached, reception is terminated as soon as those buffers already assigned to the line have been filled. An error flag is set in bit 7 of the message error record for the message (see *Appendix B*). CUTOFF also turns on bit 7 of the message error record if the input buffer is filled with identical characters (usually an indication of station malfunction).

An ERRORMSG macro may be used in the same incoming group as the CUTOFF macro to test bit 7 of the message error record and to notify the terminal operator that reception of the message has been terminated. The operator can determine if he exceeded the allowed number of characters; otherwise a station malfunction is indicated. After the CUTOFF macro has executed, processing continues through the inbuffer subgroup.

Name	*Operation*	*Operand*
[symbol]	CUTOFF	integer

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

integer

Function: Specifies the maximum number of characters allowed for each message.
Default: None. This operand must be specified.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing X' ' or XLn' ' characters must be specified.
Maximum: 65535 or a hexadecimal field two bytes in length.

Example:

```
CUTOFF 400
```

specifies that reception of the message is to be terminated after 400 characters have been entered.

NOTE: CUTOFF does not provide a precise limit to the number of characters in an incoming message (because TCAM continues to read until the buffers currently assigned have been filled).

DATETIME Macro Instruction

The DATETIME macro:

- Inserts the date, the time, or both in an incoming or outgoing message header,
- Is optional in inheader and outheader subgroups (and not permitted in any other subgroup,
- May not be specified for audio lines.

The DATETIME macro causes insertion of the date, the time, or both the date and the time in the header of an incoming or outgoing message. (If both are specified, the date is inserted first.) Seven characters are inserted for the date, if specified: a blank, the last two digits of the year, a period, and the three-digit day number. Nine characters are inserted for the time, if specified: a blank, two digits for the hour, a period, two digits for the minute, a period, and two digits for the second. If no operand is coded, both the date and the time are provided (the operands specify which is to be omitted).

Space in the header for these insertions, seven characters for the date and nine characters for the time, must be reserved by the RESERVE= operand of the DCB macro for the communication line or by the RESERVE= operand of the PCB macro for the application program if the insertions are required. After DATETIME has executed, the scan pointer is positioned at the last inserted character.

When the DATETIME macro is coded in an outgoing subgroup, the macro may operate upon the first message segment only. This is because TCAM does not maintain reserve bytes for any segment of an outgoing message except the first (see the description of the RESERVE= operand of the DCB macro).

To avoid having to specify a large first buffer, the user who wishes to insert both the date- and time-received and the date- and time-sent in the same message header may design his header so that it occupies two buffers, and then insert the incoming date and time in that portion of the header contained in the second buffer, and the outgoing date and time in that portion of the header contained in the first buffer.

The characters inserted by DATETIME are in EBCDIC code. Therefore, the DATETIME macro should not be issued *before* a CODE macro in an inheader subgroup, or *after* a CODE macro in an outheader subgroup.

DATETIME may not be specified for audio lines.

Name	*Operation*	*Operand*
[symbol]	DATETIME	[DATE={NO / YES}] [,TIME={NO / YES}]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

DATE={NO / YES}

Function: Specifies whether the date is to be inserted in a message header.
Default: DATE=YES
Format: YES or NO.
Notes: YES specifies that the date is to be inserted in the message header.

NO specifies that the date is to be omitted from the header.

TIME={NO / YES}

Function: Specifies whether the time is to be inserted in a message header.
Default: TIME=YES
Format: YES or NO.
Notes: YES specifies that the time is to be inserted in the message header.

NO specifies that the time is to be omitted from the header.

The time inserted is the time at which this DATETIME macro is executed. TCAM determines this by examining the system timer.

NOTE 1: If no operand is coded, both the date and the time are inserted in the message header.

NOTE 2: If insufficient buffer space is available (too few reserve characters), the DATETIME macro does not execute and a X'04' return code is set in register 15.

Example:
The message

NYC 0039 * (message text) EOT

is entered at the NYC terminal (NYC is the origin, 0039 the input sequence number). If the header buffer is being processed at 9:45:50 p.m. on February 6, 1970 and if the SEQUENCE macro is followed by DATETIME DATE=NO, the time is inserted in the header, which becomes

NYC 0039 21.45.50 * (message text) EOT.

If the SEQUENCE macro is followed by DATETIME, the header becomes

NYC 0039 70.037 21.45.50 * (message text) EOT.

ERRORMSG Macro Instruction

The ERRORMSG macro:

- Sends an error message when an error occurs,
- Is optional in an inmessage or outmessage subgroup of an MH (and not permitted in any other subgroup),
- May be used more than once in a subgroup.

ERRORMSG sends an error message specified by the user to a designated station when errors specified by the error mask have occurred. The bits specified by the error mask operand are compared with the setting of the bits in the message error record for this message; if specified bits in the message error record are on, the error message is sent. The message may be sent unconditionally by specifying an all-zero mask, or by omitting the *mask* operand.

The message sent to the station includes the text written by the user preceded by the header of the message in error, which is recalled from the message queue. The error message, once formatted, is placed on the destination queue for the station selected to receive the message, and is handled by the outgoing group of the MH for that queue. Therefore, unless a MSGTYPE or PATH macro is used to distinguish between different message types, the format of the header of the message in error must be compatible with the macros executed in the outgoing group handling messages routed to the station selected to receive the error message. If the MSGTYPE macro is used for this purpose, the formats of the respective headers may differ after the message-type character.

If the MSGFORM macro is not coded in the outheader subgroup of the MH handling messages for the destination station, the user must ensure that satisfactory line-control characters (such as EOT) are included in his error message.

The user may prefer to use the MSGGEN function if the message header is not required as a part of the error message. MSGGEN is faster than ERRORMSG (i.e., the user is notified of the error sooner if he uses MSGGEN), but ERRORMSG returns the header of the message in error, while MSGGEN does not.

If cancellation of an erroneous message is required, the CANCELMG macro must have been issued prior to the ERRORMSG macro. ERRORMSG may appear in inmessage and outmessage subgroups and can appear more than once in either subgroup.

NOTE: Since the header of the message in error is recalled from the destination queue, it is not possible to use the ERRORMSG macro coded DEST=DESTIN when the destination of the message in error is not known to TCAM. If a message having an invalid destination field is entered, and the destination is not corrected by the user-exit of the FORWARD macro, and no dead-letter queue is specified by the INTRO macro, then ERRORMSG cannot be used in conjunction with that message, because the message header cannot be recalled by TCAM from a destination queue.

Name	*Operation*	*Operands*
[symbol]	ERRORMSG	[mask] [,CONNECT={AND / OR}] [,DEST={destination name / opfield / ORIGIN / DESTIN}] ,DATA={message / fieldname} [,EXIT=name of routine]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

mask

Function: Specifies the five-byte bit configuration used to test the message error record for the message (the message error record is described in *Appendix B*).
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing characters must be specified. If X ' ' is used, leading zeros must be coded, If XL5' ' is used, leading zeros may be omitted.
Maximum: 16777215 or a hexadecimal field five bytes in length.
Notes: Omitting this operand or specifying an all-zero mask causes unconditional execution.

CONNECT={**AND** / **OR**}

Function: Specifies the type of logical connection to be made between the mask and the message error record.
Default: CONNECT=OR
Format: AND or OR.
Notes: AND specifies that the macro is to be executed only if *all* of the bits specified by *mask* are on in the message error record.

OR specifies that the macro is to be executed if *any* bit specified by *mask* is on in the message error record.

DEST={**destination name** / **opfield** / **ORIGIN** / **DESTIN**}

Function: Specifies the destination for the error message.
Default: In an inmessage group, DEST=ORIGIN.

In an outmessage group, DEST=DESTIN.

Format: destination name, opfield, ORIGIN or DESTIN. *destination name* is the name of a single or a process entry in the terminal table and must be enclosed in framing C' ' or CLn' ' character. *opfield* is the name of an option field defined by an OPTION macro, conforming to the rules for assembler language symbols, which contains the name of a single or process entry in the terminal table. It must not be specified with framing characters.
Notes: *opfield* is a field from two to nine bytes long, with the first byte containing the decimal length of the rest of the field.

ORIGIN specifies that the error message is to be sent to the station from which the message originated. This operand may be specified in either an inmessage or outmessage subgroup. If the originating station is not known (because it called in on a switched line and did not identify itself) the message is sent to the dead-letter queue if one is specified and is lost otherwise.

DESTIN specifies that the error message is to be sent to the destination station specified in the header of the message in error. This operand may be specified in either an inmessage or outmessage subgroup.

If an invalid destination is specified, or if DESTIN is specified in an inmessage subgroup for which no FORWARD macro has been issued previously, the message is sent to the dead-letter queue if one has been specified by the DLQ= operand of the INTRO macro. If no dead-letter queue is specified, the message is overlaid and lost.

A distribution list or a PUT process entry must not be specified as the destination of an error message.

DATA={**message** / **fieldname**}

Function: Specifies the error message.
Default: None. This operand must be specified.
Format: message or fieldname. *message* is the actual error message to be sent and must be specified within framing C' ' or CLn' ' characters. *fieldname* is the name of a location containing in its first byte a binary count of the number of characters in the message, followed by the message itself. The error message is a maximum length of 255 characters. This is exclusive of the binary count in the *fieldname* format.
Notes: If an error message is longer than a single buffer unit, one additional buffer unit is obtained and as much of the remainder of the message as will fit is placed in it. If the entire message will not fit into these two units, the remainder is truncated on the right.

EXIT=name of routine

Function: Specifies the name of a user-written routine that completes error message processing. If additional processing is needed after the standard TCAM error message processing is completed, the routine is given control after processing but before the message is sent.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: If the user provides an exit routine for ERRORMSG, TCAM automatically saves and restores registers for this routine; the user need not save registers, and may change the contents of registers 2 through 12 as he likes. However, the contents of register 13 and 14 should not be altered by the user routine. When the routine receives control, register 1 contains the address of the header buffer. Register 14 contains the return address for the calling routine. Register 15 contains the address of the entry point for the user routine. TCAM expects no return code from the user routine. The routine should return control to TCAM by a BR 14 instruction.

NOTE: When ERRORMSG is executed, only the first buffer of the message in error is retrieved from the destination queue (if the header occupies more than one buffer, that portion of the header extending beyond the first buffer is not retrieved). The actual error message is placed in that portion of the first header buffer that contains message text; the error message overlays the text. If the first buffer is entirely filled with header information, or does not contain enough space after the header to hold the entire error message, TCAM automatically assigns one extra unit to the buffer to hold as much as possible of the remainder of the message. If the entire message will not fit, the remainder is truncated on the right.

NOTE: The message is inserted in the header beginning at the current location of the scan pointer. If an ERRORMSG macro is issued in the inmessage subgroup but there is additional header information that would be recognized by the outheader subgroup, the message will overlay this data, and data will be lost for outgoing processing. If data has been inserted or removed during inbuffer or outbuffer processing, the data in the buffer will be moved either to the right or the left while the scan pointer remains fixed. Thus, when the error message is inserted at the scan pointer, data that is logically part of the header may be lost, or data beyond the header may be included as part of the header information returned with the message.

FORWARD Macro Instruction

The FORWARD macro:

- Queues messages for one or more specified destinations,
- Is required in each inheader subgroup of the MH for every station and application program that can enter messages directed to a specific destination.

FORWARD allows scanning of the destination code field in the header of each incoming message and compares the field with the names of the terminal table entries. If the destination code is valid (a matching entry is found in the terminal table), FORWARD queues the message for the specified destination or destinations. If an invalid destination code (i.e., one not appearing in the terminal table) is detected, control passes to the user routine specified by the EXIT= operand of FORWARD. If no user exit is specified, the message is queued for the station or application program specified by the DLQ= operand of the INTRO macro. If no station or application program is specified by DLQ=, and no user exit is provided, messages with invalid destination codes are overlaid and lost.

Messages may be routed to one or more destinations in the following ways:

1. To the single destination specified in the message header or named by an operand of the FORWARD macro.
2. To the distribution list specified in the message header or named by an operand of the FORWARD macro.
3. To the cascade list specified in the message header or named by an operand of the FORWARD macro.
4. To the multiple destinations specified in the message header. The destination codes may be of equal length or of varying lengths. In the case of multiple destinations, an operand specifies the end-of-address character or characters included after the last destination code in the header of each incoming message.
5. To the group entry in the terminal table specified in the message header or in an operand of the FORWARD macro.

If multiple destinations are specified in the message header, or if a distribution list is specified, once the incoming group has finished processing the message, copies are made and routed to the destination queue for each destination specified in the header or distribution list.

A FORWARD macro must be included in each inheader subgroup handling messages destined for stations or application programs; otherwise the incoming group of the MH does not know where to route the message.

If DEST= (number) or DEST=** is specified, the CODE macro must be executed prior to FORWARD unless the line code is EBCDIC.

NOTE: Care must be taken in entering a character string in a destination field to ensure that it matches a terminal-table entry. A character string entered in lower-case characters from an IBM 2770 station, for example, will not match a terminal-table entry name that is in upper-case characters.

The FORWARD macro has the following format:

Name	*Operation*	*Operand*
[symbol]	FORWARD	[DEST={destname / opfield / (number) / **(default) / PUT}] [,EOA=characters] [,EXIT=name]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

DEST={destname / opfield / (number) / PUT / **(default)}

Function: Specifies the destination for the message.
Default: DEST=**
Format: destname, opfield, (number), PUT or **. *destname* is the name of a single, group, distribution list, cascade list or process entry in the terminal table and must be specified with framing C' ' or CLn' ' characters. *opfield* is the name of a field defined by an OPTION macro containing the name of an entry in the terminal table. Framing characters must not be used. *(number)* is the number of characters in each of a list of one or more destinations. PUT is specified when the destinations of messages entered by an application program are placed by the user in an application program work area. ** specifies that there are one or more destination names of variable length in the message header.
Maximum: (number) can be a decimal field with a maximum value of 8.
Notes: opfield refers to an option field that is from 1 to 8 bytes long. If the destination name is shorter than the length of the option field, the name must be padded to the right with blanks to fill the field.

If *(number)* is specified, the destination names in the message header must all be the same length. Delimiting and embedded blanks are ignored. If this operand is specified and there is more than one destination, the EOA= operand must also be specified.

If ** is specified, delimiting blanks must be used between destination names in the header, and there may not be any embedded blanks. If this operand is specified and there is more than one destination in the message, the EOA= operand must also be specified.

DEST=PUT should be specified in the inheader subgroups of the MH assigned to an application program when the MH is to handle messages coming from an application program that has OPTCD=W coded in its output DCB macro, if the user wishes the message to go to the destination specified in the work area. For more information on specifying

the destination of a message in the application program, see the discussion of the OPTCD= operand of the output DCB macro. Use of this operand is restricted to the case just described.

If an invalid destination is specified, control passes to the user routine specified by the EXIT= operand. If no user exit is specified, the message is queued for the station specified by the DLQ= operand of the INTRO macro. If no station is specified by DLQ= and no user exit is provided, messages with invalid destination codes are overlaid and lost.

EOA=characters

Function: Specifies the character or character string used after the last station name of a multiple destination to delimit the destination field of the header.
Default: None. With DEST= coded *destname, opfield,* or PUT, specification optional. With DEST= coded *(number)* or ** and multiple destinations in the message, this operand is required.
Format: One to eight nonblank characters specified in character or hexadecimal format. If character format is specified, the field may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is specified, the field must be framed with X' ' or XLn' ' characters. *n* must be the actual length of the characters.
Notes: If this operand is specified and DEST= is coded *destname, opfield* or PUT, the operand is ignored.

EXIT=name

Function: Specifies the name of a user-written exit routine that is given control when an invalid destination is detected.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: The routine may correct the destination, provide another destination, or indicate that the message is not to be processed for a destination. If an invalid destination is provided by the user exit routine, the message is forwarded to the dead-letter queue if one is specified by the DLQ= operand of the INTRO macro; otherwise it is overlaid and lost.

If the user provides an exit routine for FORWARD, TCAM automatically saves and restores registers for this routine; the user routine need not save registers and may change the contents of registers 2 through 12. However, the contents of register 13 and 14 should not be altered. When the user routine receives control, register 1 contains the address of the header buffer. Register 14 contains the return address for the calling routine. Register 15 contains the address of the entry point for the user routine.

TCAM expects the user routine to place one of two items in register 15 before returning control:

- A return code of all zeros in register 15 means that the user routine was unable to provide a satisfactory destination for this message. In this case, the message is forwarded to the dead-letter queue or is not processed for any destination if no dead-letter queue is provided.
- Register 15 may contain the main-storage address of a field set up by the user and consisting of a length byte followed by the name of a valid single, group, distribution list, cascade list or process entry in the terminal table. The length byte must contain, in binary form, the number of bytes in the rest of the field. TCAM assumes that the specified name is the destination of the message. The field must be padded to the right with blanks to the length of the longest entry.

The user routine should return control to TCAM via a BR 14 instruction.

This operand is ignored when DEST=PUT is specified.

NOTE: In the case of multiple-buffer headers, a destination must be determined for the first header buffer. This can be ensured in one of two ways as the first header and the subgroup are designed:

1. If the destination is specified by the macro operand, the FORWARD macro must occur sufficiently early in the subgroup that it acts upon the first header buffer.
2. If the destinations are specified in the header rather than by the macro operand, the first destination must be completely contained within the first buffer. For buffered terminals, the first destination must appear in the first hardware buffer or the first MCP buffer, whichever is smaller.

If the second condition is not met, TCAM assumes an invalid destination has been specified and branches to the user exit, if provided. If no user exit is provided, or if the first condition is not met, the message is routed to the dead-letter queue, or is overlaid and lost, if no dead-letter queue is provided.

HOLD Macro Instruction

The HOLD macro:

- Suspends transmission to a station,
- Is optional in the inmessage and outmessage subgroups.

HOLD suspends transmission of output messages to a station either for a time interval or until the messages are released by a RESMXMIT operator command or by an MRELEASE macro issued in an application program. HOLD may be requested unconditionally by specifying an error mask of zero or by omitting the mask, or conditionally, in which case the error mask specified in the first operand is compared to the message error record assigned to the message; if specified errors are detected, transmission is suspended. A station that cannot accept messages because of the effect of a HOLD macro is said to be *intercepted*. For a discussion of holding, see the section *TCAM's Hold/Release Facility*.

NOTE 1: A station whose destination queue is located in main storage with no disk backup may not be intercepted; the HOLD macro is ignored in this case.

NOTE 2: Suspension of transmission begins with the message following that which causes the HOLD macro to execute (since the outmessage subgroup does not execute until after the message has been sent). However, when the station is released, the message that caused HOLD to execute is retransmitted.

NOTE 3: If an initiate mode message is sent to a held terminal, the message will revert to standard transmission (rather than initiate transmission). However, it will be queued on the highest priority queue and be transmitted normally thereafter.

NOTE 4: If the HOLD macro is executed in the outmessage subgroup for a lock response, the lock is not broken, the terminal is not held, and the message will be retransmitted immediately (i.e., it will be sent twice). This can result in an infinite loop if the condition for the HOLD is permanent and the line or terminal is inoperative. If a terminal is held by an operator command while in lock mode, or if lock is initiated while the terminal is held, all lock responses will be sent as if the terminal were not held. No other messages will be sent, however, until the terminal is released.

Name	*Operation*	*Operands*
[symbol]	HOLD	[mask] [,RELEASE] [,INTVL=integer] [,CONNECT={AND}] {<u>OR</u> }

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

mask

Function: Specifies the five-byte bit configuration used to test the message error record for the message (the message error record is described in *Appendix B*).
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing characters must be specified. If X' ' is used, leading zeros must be coded. If XL5' ' is used, leading zeros may be omitted.
Maximum: 16777215 or a hexadecimal field five bytes in length.
Notes: Omitting the operand or an all-zero mask causes unconditional execution.

If queuing is by line and a nonzero mask is specified, the mask must include the test for the "terminal inoperative" bit of the message error record.

RELEASE

Function: Specifies that transmission to the station is to be suspended until either a RESMXMIT operator command is issued for the station, or until an MRELEASE macro is issued for the station in an application program.
Default: None. Specification optional.
Format: RELEASE
Notes: If this operand is omitted and INTVL= is also omitted, RELEASE is assumed. If both RELEASE and INTVL= are coded, RELEASE prevails.

INTVL=integer

Function: Specifies the number of seconds that transmission to the station is to be suspended.
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing X' ' or XLn' ' characters must be specified.
Maximum: 65535 or a hexadecimal field two bytes in length.
Notes: At the end of the specified period, transmission to the station is automatically resumed. If this operand is omitted, RELEASE is assumed. If both RELEASE and INTVL= are coded, RELEASE prevails.

CONNECT= {AND / **OR**}

Function: Specifies the type of logical connection to be made between the mask and the message error record.
Default: CONNECT=OR
Format: AND or OR.
Notes: AND specifies that the macro is to be executed only if *all* of the bits specified by *mask* are on in the message error record.

OR specifies that the macro is to be executed if *any* bit specified by *mask* is on in the message error record.

NOTE: The TCAM checkpoint/restart facility permits restart of a TCAM system after system closedown or failure. If the system fails or is closed down while the station is intercepted, when the system is restarted by a warm start or continuation restart (defined in the discussion of the checkpoint/restart facility) the interception will still be in effect, but the INTVL= operand will no longer apply; transmission will be suspended until a RESMXMIT operator command or MRELEASE macro causes transmission to be resumed.

INITIATE Macro Instruction

The INITIATE macro:

- Sends message segments to their destination as soon as possible after they are received at the destination queue,
- Is optional in an inheader subgroup of an MH.

The INITIATE macro sends the segments of a message from a destination queue to the destination as soon as possible after they are placed on the queue. (Normally, segments are not sent to the destination until after the complete message has been placed on the queue.) For information on when messages destined for stations on the same line are sent out relative to each other, see *Message Priority and Queuing* in the chapter *Defining Terminal and Line Control Areas.* The destination may be either a station represented by a single or group entry in the terminal table, or an application program represented by a process entry in the terminal table. This function may be specified conditionally, based on the appearance of a specified character in the message header, or it may be specified unconditionally.

When the first segment of a message processed by INITIATE arrives on a destination queue, it is treated as if it were a complete message having the highest priority on the queue. If the destination queue was created by a TERMINAL macro, as soon as a line to the destination station is available, TCAM begins sending that portion of the message that has arrived at the destination queue. No other message may be sent on the line until this entire message has been transmitted. If the destination queue was created by a TPROCESS macro, then each message segment is sent to the application program as soon as possible after it is enqueued.

If a message is sent to a station for which messages are being held (see the description of the HOLD macro), the message reverts to normal transmission mode rather than remaining in initiate mode. The message is queued on the highest-priority queue and is transmitted to its destination after the station is released for accepting messages. Once the station is released from its hold condition, TCAM resumes transmitting message segments to the destination via the initiate mode as described above.

The function provided by the INITIATE macro might be used as an early notifiction to a destination station that a very long message is being received by the computer, handled and routed to that destination.

If a message has multiple destination codes specified in the header, the INITIATE function is performed only for the first destination. Sending to the remaining destinations will occur only after the complete message has been placed on the destination queue.

If static deallocation of buffers is specified (i.e., if the PCI= operand of the line group DCB macro is coded PCI=N, and the incoming message contains no EOB or ETB control characters), the only effect of INITIATE is to give the message a priority higher than that of any other message on the destination queue.

A message to be transmitted in initiate mode must contain no EOB, ETB, or ETX line-control characters, either when it is received by the computer or when it is sent from the computer, otherwise, hardware errors may result.

Block checking is not performed for messages transmitted in initiate mode. Since buffered terminals require this checking, the INITIATE macro should not be executed for messages entered by buffered terminals.

Name	*Operation*	*Operand*
[symbol]	INITIATE	[conchars[,BLANK={char / NO / YES}]]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

conchars

Function: Specifies the character or character string that, if found in the header as the next nonblank field, causes execution of the function.
Default: None. Specification optional.
Format: One to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters.
Notes: If this operand is omitted, the INITIATE function is performed unconditionally. If the next field in the header does not match this operand, the function is not performed.

BLANK={char / NO / YES}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES

Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: This operand is meaningless unless the *conchars* operand is also specified.

YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars.*

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

Example:

```
INITIATE C'&'
```

causes the INITIATE function to be executed whenever the & character appears as the next nonblank character in the message header.

NOTE: In the case of multisegment headers the INITIATE function must apply to the first segment of the message. This is ensured by designing the message header so that the control characters appear in the first segment.

LOCK Macro Instruction

The LOCK macro:

- Connects one station on a line to an application program to await the response to an inquiry message entered by the station,
- Holds the connection for a single message or for an extended period,
- Is optional in an inheader subgroup (and not permitted in any other),
- Should not be used to specify extended lock mode with IBM 2740 Model 2 terminals,
- Is suggested for audio terminals.

LOCK keeps the connection between a station and an application program, as specified in a message header or by a FORWARD macro, for a period of time not less than the duration of a message and its response. A station connected in this manner is said to be in *lock mode.* The application program to which a station is locked depends upon the destination in the header or that specified by a FORWARD macro. If the destination is not an application program, the station is not placed in lock mode.

NOTE: LOCK does not execute if the station that entered the message being handled is a buffered station whose TERMINAL macro specified a buffer delay (via the BFDELAY= operand). In this case, a return code of X'00000004' is passed in register 15 by TCAM's lock routine.

For a description of the lock function, see *TCAM's Inquiry/Rapid Response Facility* in the chapter *Writing TCAM-Compatible Programs.*

The LOCK macro has the following format:

Name	*Operation*	*Operand*
[symbol]	LOCK	{EXTEND / MESSAGE} [,conchars[,BLANK={YES / NO / char}]]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

{EXTEND / MESSAGE}

Function: Specifies the type of lock mode required.
Default: MESSAGE
Format: EXTEND or MESSAGE.
Notes: EXTEND specifies that the station transmitting the message is to be placed in lock mode until it has no more messages to transmit or until an UNLOCK macro is executed.

MESSAGE specifies that the station transmitting the message is to be placed in lock mode for the duration of the message and its response, and that the line is to be freed once the response has been sent.

conchars

Function: Specifies the character or character string that, if found in the header as the next nonblank field, causes execution of the function.
Default: None. Specification optional.
Format: One to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters.
Notes: If this operand is omitted, the LOCK function is performed unconditionally. If the next field in the header does not match this operand, the function is not performed.

For a station in extended lock mode, control characters are meaningful only for the header of the message being processed at the time the station is placed in lock mode. The LOCK macro does not examine headers of messages entered by a station already in extended lock mode for control characters.

BLANK={YES / NO / char}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header which is being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES
Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: This operand is ignored unless the *conchars* operand is also specified. YES specifies that the EBCDIC *blank* character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars*.

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

NOTE: For a station in extended lock mode, control characters are meaningful only in the header of the message being processed at the time that the station is placed in lock mode. The LOCK macro does not examine the headers of messages entered by a station already in extended lock mode for control characters.

LOCOPT Macro Instruction

The LOCOPT macro:

- Provides access to fields in the option table,
- Is optional in inheader, inbuffer, outheader, and outbuffer subgroups (and not permitted in any other).

LOCOPT enables the user to obtain the address of any option field for the appropriate terminal table entry. The address of the desired field or a not-found indicator is placed in a user-specified register. A user-written routine may then examine and modify the contents of the option field. If specified in the incoming group, LOCOPT accesses option fields for the originating station; if specified in the outgoing group, LOCOPT accesses option fields for the destination station. If specified in an MH handling messages to or from an application program, LOCOPT locates the option fields in the process entry for the queue to which the GET or READ is directed (if LOCOPT is issued in the outgoing group), or the fields in the process entry for the queue to which the PUT or WRITE is directed (if LOCOPT is issued in the incoming group). LOCOPT may be used only for option fields for stations or application programs using the MH in which LOCOPT is issued.

Name	*Operation*	*Operand*
[symbol]	LOCOPT	opfield,{(register) / (15)}

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

opfield

Function: Specifies the name of the option field whose address is desired.
Default: None. This operand must be specified.
Format: Must be the name of an option field as defined by an OPTION macro.
Notes: If the option field is not found, LOCOPT does not execute and a X'04' return code is set, unless the default register 15 is used for the address of the option field, in which case register 15 contains a fullword of zeros on return.

{(register) / (15)}

Function: Specifies the register into which the address of the desired option field is to be placed.
Default: (15).
Format: A decimal register 2 through 11 or 15, enclosed in parentheses.

LOG Macro Instruction

The LOG macro:

- Enables the user to log complete messages or message segments,
- Is optional in any subgroup of an MH.

LOG enables the user to maintain a record of incoming or outgoing message traffic on a sequential medium. Message segments or full messages, as determined by the placement of

LOG macros in an MH, are placed on an output device. The various types of logs, and the corresponding MH subgroups in which LOG appears are:

1. Incoming header segments only (inheader)
2. All incoming segments (inbuffer)
3. Complete incoming messages (inmessage)
4. Outgoing header segments only (outheader)
5. All outgoing segments (outbuffer)
6. Complete outgoing messages (outmessage).

When LOG is specified in an inbuffer or outbuffer subgroup, segments are logged in the sequence in which they are handled by the Message Handlers. In this case, segments of different multi-segment messages handled at about the same time are likely to be intermixed on the logging medium. When segments are logged, their buffer prefixes are logged with them. The 12-byte control area connected with each buffer unit is not logged.

LOG may appear at any point in an MH subgroup in which it is used. However, the results of any alteration of segments or messages by macros preceding LOG in the subgroup will appear in the log. For example, if LOG is preceded by DATETIME, a logged header segment will contain the date or time, as specified in DATETIME, depending on the location of the date and time in a multi-segment message.

LOG may be specified in any subgroup of an MH and may be used more than once in a subgroup if desired. The message log may be maintained on any available output medium. The user must supply, define, and open the message log data sets. For each log data set used to log complete messages, a logtype entry in the terminal table must be defined by a LOGTYPE macro (this is not necessary if only segments are logged). For information on specifying the message log data set, see *Defining the MCP Data Sets.*

NOTE: When logging segments after a FORWARD macro with multiple destinations, the last character of the first destination is overlaid with an unprintable character. This byte will be restored at the inmessage subgroup and thus will appear if messages are logged.

Name	*Operation*	*Operand*
[symbol]	LOG	{dcbname / typename}

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

{dcbname / typename}

Function: Specifies the name of the data control block or the logtype entry used for logging.
Default: None. This operand must be specified.
Format: dcbname or typename. *dcbname* is the name of the data control block for the message log data set, and is used if the macro is specified in the inheader, inbuffer, outheader or outbuffer subgroup. *typename* is the name of a logtype entry in the terminal table, and is used if the macro is specified in the inmessage or outmessage subgroup.
Notes: If *dcbname* is specified and does not match the name of a valid data control block, or if *typename* is specified and does not match the name of a logtype entry in the terminal table, the LOG macro does not execute, and a return code of X'04' is set in the low-order byte of register 15.

MSGEDIT Macro Instruction

The MSGEDIT macro:

- Inserts specified characters at specified locations in a message,
- Deletes specified characters from a message, and
- Replaces deleted characters with other characters, or contracts remaining data to fill the gap caused by deletion,
- Dynamically allocates buffer units to contain data inserted in message segments,
- Is optional in inheader, inbuffer, outheader, and outbuffer subgroups, and may not be coded in any other subgroup.

The MSGEDIT macro allows the user to edit incoming and outgoing messages from a Message Handler. Each editing operation performed by MSGEDIT falls into one of two categories: it is either an insertion or a removal.

An *insert operation* is one in which specified characters are inserted at a specified point in a message, with no characters being deleted in the operation. The operands of MSGEDIT allow characters to be inserted

- at a single point in a message;
- at a specified offset from the beginning of each message segment;
- whenever a certain character string appears in a message;
- after every *n* bytes of message date, where *n* is a number specified by the user.

The inserted data may consist of a single character, an ordinary character string, or a string of identical characters. If the MSGEDIT macro is issued in an inheader or outheader subgroups, the insert operation is performed only for a single segment of a message. This is usually the first segment, but may be a subsequent segment if the message has a multiple-buffer header and the MSGEDIT macro is issued in a portion of the subgroup that is processing header fields in the second or subsequent segments. (The manner in which inheader and outheader subgroups are executed for multiple-buffer headers is described in the chapter *Designing the Message Handler*.) If the MSGEDIT macro is issued in an inbuffer or outbuffer subgroup, the insert operation is performed for each segment in the message. The insert function might be used to add a new destination name to the destination field in a message header, or to insert idle characters into an outgoing message going to a terminal with a printer requiring such characters to prevent "printing on the fly" during a carriage-return operation. For other uses, see the examples below.

A *remove operation* is one in which a specified character string is removed from a message. The user may specify that the character string be replaced with another character string, or that data remaining in the segment after removal be contracted to fill the gap left by the deleted data. The user may remove

- a single character string;
- a specified character string whenever it appears;
- a specified number of bytes of data whenever a certain character string appears;
- the data located in a specified section of a buffer.

In any of the above cases, the user may replace the deleted data with other data, or he may specify that data following the deleted data in a message segment be moved left to fill the gap left by the deleted data. If a substituted character string is longer or shorter than the deleted character string, TCAM automatically spreads or contracts the data remaining in the buffer to "fit" the new string; buffer units are allocated as needed to accommodate the new data. If MSGEDIT is coded in an inheader or outheader subgroup, data is removed from only a single header segment of a message. If MSGEDIT is coded in an inbuffer or outbuffer subgroup, data is removed from all message segments. The remove function might be used to delete a destination from the destination field in the message header, to substitute one destination name for another in the header, to remove unnecessary data from an outward-bound message, or to replace a specified character with a logical-record delimiter that is recognized by application-program GET macros.

If the buffer containing a message segment is not long enough to accommodate additional data inserted by a MSGEDIT macro, additional buffer units are automatically added to the buffer as needed. Empty units at the end of a buffer are automatically deallocated when the buffer is passed to an INMSG or OUTMSG macro; deallocated units are returned to the available-unit queue.

Up to 31 separate insert and remove operations may be specified by issuing a single MSGEDIT macro having up to 31 groups of positional operands. However, assembler language restrictions on the length of a macro operand also apply.

The MSGEDIT macro operand field consists of from one to 31 groups of four operands each, and a single keyword operand that is coded as the last operand of the macro. Each group of positional operands is enclosed in parentheses, and each specifies a single insert or remove operation (which may, however, entail multiple insertions or deletions). If the

user wishes to perform many insert or remove operations on his messages, he may either code a single MSGEDIT macro having many groups, or he may code several MSGEDIT macros, each performing one or two insert or remove operations.

A single MSGEDIT macro with five groups executes more rapidly than would five MSGEDIT macros, each having one of the groups. However, certain restrictions that apply to a MSGEDIT macro having several groups are not applicable when several MSGEDIT macros having one group each are used instead (these restrictions are discussed below in the description of the MSGEDIT operands). Thus, the tradeoff to be considered when deciding whether to specify one MSGEDIT macro with several groups of operands or several MSGEDIT macros with one group each is between speed of execution and flexibility.

Each group contains an AT operand, which specifies where, in a buffer, an insert or remove operation is to begin. The order in which operations are performed depends upon the relative locations of the character strings specified by the AT operand in each group. The function specified by the group whose AT operand appears first in a particular message segment is performed first for that segment, the function specified by the group whose AT operand appears second is performed second, etc.

NOTE 1: If end-of-block checking is specified for the message handler, the MSGEDIT macro may not be used in an incoming group to expand or contract the amount of data in the buffer. If the MSGEDIT macro is used in this manner and an error occurs in transmission, the retransmission of the segment will result in duplicated data if the buffer is contracted and lost data if it is expanded. This restriction does not apply if static allocation and deallocation of buffers is specified for receive operations (by coding N as the first suboperand of the PCI= operand of the line group DCB macro).

NOTE 2: When multiple groups of positional operands are coded for a MSGEDIT macro, rather than multiple MSGEDIT macros each with a single group, data inserted by one operation is not considered to be part of the message segment when another operation is being performed. For example, if one group caused a B character to be inserted after every A character in the message, and another group of the same MSGEDIT macro specified that a C character be inserted after every B character in the message, no C character would be inserted after a B character that was itself inserted as a result of an A character being encountered in the message segment by the MSGEDIT macro.

Insertion or removal of data using a MSGEDIT macro always results in a movement of data in the buffer. Even when a MSGEDIT macro specifies only a single remove operation and the replacement string is equal in length to the character string being replaced, movement of data occurs (though in this case the result of the data movement would be that the replacement string occupies the space originally occupied by the deleted string). As a rule, when a MSGEDIT macro operates on any data in a buffer, all of the data between the characters affected by the first insert or remove operation and the end of the buffer is shifted once by means of MVC instructions issued internally by TCAM. No data is shifted more than once per MSGEDIT macro, regardless of the number of operations specified in the macro.

The MSGEDIT macro has certain limitations:

1. When issued in an inheader or outheader subgroup, MSGEDIT acts only upon one header segment of messages having multiple-buffer headers. The segment acted upon is the one being processed by the inheader or outheader subgroup at the time MSGEDIT is executed. Moreover, a MSGEDIT macro issued in an inheader or outheader subgroup assumes that the header occupies the entire segment being operated upon. Thus, if a MSGEDIT macro in an inheader subgroup specifies that NYC is to replace BOS whenever the latter character string occurs in the header, and if the header ends midway through the first message segment, BOS will be replaced if it appears in the second half of the segment, even though it is outside of the header.
2. A character string to be removed may not extend across segments; the delimiters for both ends of the character string must be located in the same buffer.
3. Any character string in an operand specified in character format rather than as hexadecimal data cannot include a comma or a right parenthesis. If the character field requires the use of these characters, the field must be specified in hexadecimal format.

4. The user must beware of performing MSGEDIT functions that either add or remove data to the left of the scan pointer while he is performing sequential processing of header fields. Because the scan pointer points to a particular physical location in the buffer, rather than to a particular character; addition of data to the left of the scan pointer results in the shifting of the original scan pointer to the left. The following example illustrates the possible problem resulting from improper placement of a MSGEDIT macro in the message handler:

```
SETSCAN C'X'
ORIGIN 5
MSGEDIT ((I,C'INSERT',1))
FORWARD DEST=5,EOA=*
```

After the SETSCAN and the ORIGIN macros are executed, the buffer might look like this:

```
prefix  X TERMA  TERMB  TERMC * message data
          ↑
          scan pointer
```

After the MSGEDIT macro executes, the buffer looks like this:

```
prefix  INSERT  X  TERMA TERMB TERMC * message
                ↑
                scan pointer
```

When the FORWARD macro executes, the origin (TERMA) will be considered to be the first destination (TERMB). To avoid such problems, the user may follow these two guidelines:

1. Perform as many of the MSGEDIT functions as possible in an INBUF or OUTBUF subgroup rather than in INHDR or OUTHDR.
2. Perform all MSGEDIT functions that affect header fields either before all sequential processing of header fields begins, or after all sequential processing of header field has been completed. Examples are:
   a.
   ```
   MSGEDIT ((I,C'INSERT',1))
   SETSCAN C'X'
   ORIGIN 5
   FORWARD DEST=5,EOA=*
   ```
   b.
   ```
   SETSCAN C'X'
   ORIGIN 5
   FORWARD DEST=5,EOA=*
   MSGEDIT ((I,C'INSERT',1))
   ```

NOTE: MSGEDIT adjusts the scan pointer backwards for the user for one special case. This is a remove (or replace) function specifying the scan pointer itself as the TO operand. Examples of this are:

```
MSGEDIT ((R,,25,SCAN))

MSGEDIT ((R,C'INSERT',25,SCAN))
```

In these examples, if the remove or replace function results in the deletion of more bytes than exist between the scan pointer and the end of data in the buffer after the macro executed, the scan pointer would, if not adjusted, erroneously point beyond the end of the data in the buffer and prevent any subsequent sequential processing. Therefore, in these cases, the scan pointer is moved backward a distance equal to:

a. The length of the data removed, or
b. The length of data removed less the length of data inserted.

The MSGEDIT macro is far and away the most complex TCAM functional macro. The user is cautioned that he may have to read the following description several times before he understands how to code the macro. Several examples follow the macro description.

The MSGEDIT macro instruction has the following format:

Name	*Operation*	*Operands*
[symbol]	MSGEDIT	((group1),(group2),...)[,BLANK={char / NO / YES}]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

((group1),(group2),...)

Function: Each group specifies a single insert or remove function.
Default: None. At least one group must be specified.
Format: Each group contains (function,data,AT,TO) operands. They must be provided in the order shown, enclosed in parentheses, and separated from each other by a comma.
Maximum: A maximum of 31 groups may be coded.
Notes: Due to the complexity of the macro, the operands are explained individually below.

The structure of each group of positional operands is as follows:

Function operand	*Data operand*	*AT operand*	*TO operand*
{I / R[A] [T]} ,	[characters / (hexform,n) / DELIMIT / CONTRACT],	[characters / offset / (integer,optfield) / SCAN],	[characters / offset / SCAN / (count) / (0)] ,...

function operand
{I / R[A] [T]}

Function: Specifies whether an insert or remove function is to be performed and, if a remove function, whether the characters delimiting the beginning and the end of removal are themselves to be removed.
Default: None. This operand must be specified.
Format: I, R, RA, RT, RAT or RTA.
Notes: I specifies that an insertion function is to be performed. The data specified in the data operand is the data inserted in the message.

R specifies that a remove function is to be performed; any data specified by the AT operand and the TO operand is to be removed from the message and replaced with the data specified by the data operand. If no data is specified by the AT operand or by the TO operand, MSGEDIT removes one byte of data beginning at the location currently designated by the scan pointer. If no data is specified by the data operand, data remaining in a buffer after a deletion is contracted to fill the space left by the deleted data.

A specifies that removal is to begin with the first character of the character string specified in the AT operand; in this case, if replacement data is specified in the data operand, the first byte of replacement data is inserted in the space occupied originally by the first byte of the character string specified by the AT operand. If A is omitted, removal and replacement begin with the character immediately following the last character in the string specified in the AT operand. If A is coded in a group, a character string should be coded as the AT operand; otherwise, MSGEDIT removes one byte of data beginning at the location currently designated by the scan pointer and proceeds to the next group, if any, to accomplish the next insert or remove function.

T specifies that removal is to end with the last character of the string specified in the TO operand; if T is not coded, the character immediately preceding the first character of the string specified by the TO operand is the last character removed. If T is coded in a group, a character string should be specified as the TO operand; otherwise, MSGEDIT removes one byte of data beginning at the location currently designated by the scan pointer and proceeds to the next group, if any, to accomplish the next insert or remove function.

data operand

characters
(hexform,n)
DELIMIT
CONTRACT

Function: If this is an insert function, specifies the data to be inserted in the message. If this is a remove function, specifies either the data to replace the characters removed from the message or specifies that the data remaining in a buffer after deletion is to be contracted to fill the space originally occupied by the deleted data.
Default: CONTRACT
Format: characters, (hexform,n), DELIMIT or CONTRACT.

characters may be one to eight nonblank characters in character or hexadecimal format. If character format is used, framing C' ' or CLn' ' characters must be used. If hexadecimal format is used, framing X' ' or XLn' ' characters must be specified.

(hexform,n) must be coded within parentheses. *hexform* is a single character in hexadecimal or character format surrounded by framing X' ' or C' ' characters. *n* is a decimal integer and must not be framed.
Maximum: *n* may have a maximum length of the length of one buffer unit.
Notes: *characters* in an insert operation specifies the character string to be inserted into the message. In a remove operation, *characters* specifies the character string that is to replace the deleted data. If messages are to be translated, inserted characters should be in EBCDIC; if they are not to be translated, inserted characters should be in terminal transmission code.

(hexform,n) specifies that the single character represented by *hexform* is to be inserted the number of times indicated by *n*. The inserted characters will be contiguous; if this is a remove operation, they will replace the deleted data. This operand may be used to insert idle characters in outgoing messages.

DELIMIT is valid only if the function operand specifies a remove function. DELIMIT specifies that the character in the RECDEL= operand of the TPROCESS macro whose name is entered as this message's destination is to replace the character string delimited by the AT and TO operands. This character is recognized by the application program's GET macro as the delimiter of a variable-length record. The MSGEDIT macro in which this operand is coded is normally located in the outbuffer subgroup of the MH for the application program or inbuffer subgroup for the line over which the message is received. If MSGEDIT is located in an inheader subgroup, only a single header segment is scanned for the character to be replaced. The destination queue must be identified by means of a FORWARD macro before the MSGEDIT macro is issued. If the destination of this message is not an application program, the MSGEDIT group containing DELIMIT does not execute.

CONTRACT is valid only if the function operand specifies a remove function. CONTRACT specifies that after the appropriate data has been deleted from a message segment, succeeding characters in the buffer are to be moved to overlay deleted characters. If contraction results in one or more empty units at the end of the buffer, these are released when the segment leaves the incoming or outgoing group of the MH.

If the function operand specifies an insert function and if CONTRACT is coded (or if the data operand is omitted), this MSGEDIT macro does not execute, and control passes to the next instruction in the MH.

AT operand

characters
offset
(integer,opfield)
SCAN

Function: If an insert function is being performed, specifies the location at which the insertion is to be made. If a remove function is being performed, specifies the location of the beginning of the string to be removed.
Default: SCAN
Format: characters, offset, (integer,opfield), or SCAN.

characters specifies one to eight nonblank characters in either character or hexadecimal format. If character format is used, the string must be framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters.

offset is a decimal integer specified without framing characters.

(integer,opfield) must be coded within framing parentheses. *integer* may be specified either in decimal or hexadecimal format. If hexadecimal format is used, the value must be coded within framing X' ' or XLn' ' characters. *opfield* is the name of a halfword option field defined by an OPTION macro.
Maximum: For *offset,* 65535. For *integer,* 65535, or a hexadecimal field two bytes in length.
Notes: If this is an insert function, *characters* specifies a string, immediately following which the data specified in the data operand is to be inserted. If the MSGEDIT macro is included in an inheader or outheader subgroup, the specified data is inserted each time this string is encountered in the message header. If the MSGEDIT macro is issued in an inbuffer or outbuffer subgroup, the specified data is inserted each time this string is encountered anywhere in the message.

If this is a remove function, *characters* specifies a string that delimits the beginning of the data to be removed. If the A suboperand of the function operand is included, removal begins with the first character of this string; if A is not included, removal begins with the character immediately following the last character of this string. If A is coded in the function operand and the TO operand is coded (0) or is omitted, only the string specified in the AT operand is removed. If the MSGEDIT macro is included in an inheader or outheader subgroup, removal occurs each time the character string is encountered in the message header. If the macro is issued in an inbuffer or outbuffer subgroup, removal occurs each time the character string is encountered in the message.

If *character* is coded, either *characters* or *(count)* should be specified as the TO operand. If SCAN is specified as the TO operand, TCAM assumes a count of zero has been specified for TO. If an offset is specified for the TO operand, TCAM assumes that the offset is a count.

If *characters* is coded, the entire string must be located within a single buffer. If more than one group of operands is included in this macro, the AT operand for each group must be specified as *characters,* and each character string specified as an AT operand must begin with a different character.

If this is an insert function, *offset* specifies the number of bytes beyond the buffer prefix immediately following which the first character specified in the data operand is to be inserted. If this is a remove function, *offset* specifies the number of bytes beyond the prefix immediately following which deletion of data is to begin.

If the MSGEDIT macro is specified in an inheader or outheader subgroup, *offset* applies to a single header segment only, and insertion or deletion of data occurs only once. If the macro is coded in an inbuffer or outbuffer subgroup, data is inserted or deleted at the specified offset in every segment of the message. If this is an insert operation and an offset of 2 is specified, the first character inserted will immediately follow the contents of the second byte beyond the buffer prefix. If this is a remove function and an offset of 2 is specified, the first byte whose contents are removed from a segment will be the third byte beyond the buffer prefix.

(integer,opfield) specifies that the data coded for the data operand is to be inserted after every number of bytes specified by *integer.* If *integer* is 20, for instance, the data specified in the data operand is inserted after every 20 bytes of message. Insertion will occur in both the header and text. *opfield* is the name of an option field assigned to the origin (if MSGEDIT is coded in the incoming group) or to the destination (if MSGEDIT is coded in the outgoing group). The option field must be initialized via the OPDATA= operand of the TERMINAL or TPROCESS macro (it may be set to a halfword of zero).

(integer,opfield) coded as the AT operand has the following restrictions:

- I must be coded as the function operand.
- This MSGEDIT macro may be coded in an inbuffer or outbuffer subgroup only.
- Only one group of positional operands may be specified.
- *characters* or *(hexform,n)* must be specified for the data operand.

SCAN specifies that insertion or deletion is to begin with the character immediately following the byte at which the scan pointer is currently pointing (see the description of the scan pointer in *Designing the Message Handler*). This operand may be specified only when the macro is issued in an inheader or outheader subgroup.

TO operand

[**characters**
offset
SCAN
(count)
(0)]

Function: For remove functions only, specifies the end of the character string to be deleted.
Default: (0)
Format: characters, offset, SCAN, (count) or (0).

characters specifies a one to eight byte field in either character or hexadecimal format. If character format is used, framing C' ' or CLn' ' characters must be specified. If hexadecimal format is used, framing X' ' or XLn' ' characters must be specified.

offset specifies a decimal integer, coded without framing characters. *(count)* must be coded within its framing parentheses, and is a decimal integer specified without framing characters.
Maximum: Both *offset* and *(count)* have a maximum value of 65535.
Notes: characters indicates the location of the last character to be deleted. If the T suboperand of the function operand is coded, deletion ends with the last character of the string specified here; otherwise, deletion ends with the character immediately preceding the first character of the string. The entire string must be located in the buffer that contains the delimiter specified by the AT operand, since deletion must begin and end in the same buffer. If both the AT and the TO operand specify character strings, TCAM assumes that the first byte of the TO string is to the right of the last byte of the AT string.

offset specifies an offset from the beginning of the data in a message segment; this offset defines the end of the string to be deleted in this operation. If the offset is 20, for instance, the character occupying the twentieth byte from the beginning of data in the buffer is the last character deleted. The offset must specify a byte that is in the same buffer as, and either in the same position as or to the right of the first byte of data removed (as specified by the AT operand); each deletion must begin and end in the same buffer. If the offset specified by the TO operand is identical with the offset specified by the AT operand, the single character located at this offset is removed. If the offset is beyond the end of the buffer, data will be deleted to the end of the buffer.

If this MSGEDIT macro is specified in an inheader or outheader subgroup, *offset* applies to a single header segment only and deletion occurs only once. If the macro is coded in an inbuffer or outbuffer subgroup, data is deleted from each segment.

SCAN specifies that the character indicated by the current position of the scan pointer is to be the last character deleted in this remove operation. This operand may be coded only in a MSGEDIT macro issued in an inheader or outheader subgroup. If SCAN is coded for both the AT and the TO operand, and R is specified in the function operand, the single character located at the current position of the scan pointer is deleted.

(count) and its default value (0) specify the number of bytes of data to be deleted, starting with the byte immediately following the AT operand. If the AT delimiter is a character string and if A is coded in the function operand, the amount of data removed is equal to the sum of the number of characters in the AT delimiter string plus the number of bytes specified by *count.* If the integer specified by *count* is greater than the number of bytes between the AT delimiter and the end of the buffer, all characters between the AT delimiter and the end of the buffer are deleted. A count of zero indicates that no data is to be deleted (except for the characters in the AT delimiter if A is coded in the function operand); if the TO operand is omitted, a count of 0 is assumed. If A is coded in the function operand and a string is coded in the AT operand, the string is removed each time it is encountered if (0) is coded or if no TO operand is specified.

BLANK={ **char** / **NO** / **YES** }

Function: This operand specifies whether EBCDIC blank characters are to be ignored when encountered in searching the message for a field, or whether blanks are to be considered part of the field when encountered. If EBCDIC blanks are to be counted when found, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in searching the message for a field.
Default: BLANK=YES.

Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in a message. For example, if BLANK=YES is coded and an eight-byte field is being acted upon by this macro, a blank appearing in the fifth will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

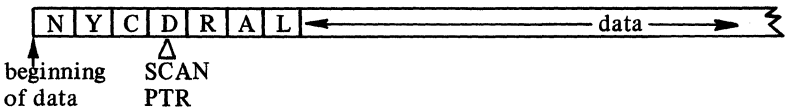
NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the message.

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header. That is, the macro automatically skips over the character without checking it. If BLANK=char is coded and *char* is not the EBCDIC blank, the EBCDIC blank is treated like any other character.

Restriction 1: Deletion must always begin and end in the same buffer. The entire character string to be deleted must reside in a single buffer.

Restriction 2: With one exception (when both the AT and the TO operands are coded as SCAN), the first byte of a string of data to be removed, as determined by the AT operand, must be to the left of, or in the same position as, the last byte of the string of data to be removed, as determined by the TO operand. See the examples following.

NOTE: The first character in a character string to be deleted, as specified by the AT operand, must not be to the right of the last character of the character string, as specified by the TO operand. If both operands specify the same byte, that byte only is removed. As an example, consider the following initial portion of a buffer, with the scan pointer pointing at D:



A MSGEDIT macro coded

```
MSGEDIT ((R,CL3'BOS',SCAN,4))
```

would result in the character D being replaced with the string BOS in the buffer.

A MSGEDIT macro coded

```
MSGEDIT ((R,CL3'BOS',CL1'D',CL3'RAL'))
```

would result in BOS inserted after D; this macro says to remove the character between D and R and replace it with BOS. Since there is no character between D and R, none is removed, but BOS is still inserted.

Examples:
MSGEDIT is a complex macro, capable of performing many functions. In this section, some of the more common functions of MSGEDIT are discussed and illustrated with examples.

Insertion of a single character string after a specified field in a header buffer: The following MSGEDIT macro might be coded in an inheader subgroup to add the destination RAL to the list of destinations specified in the message header. Assume that the last destination specified in the header is NYC, and that DEST=(3) is coded in the FORWARD macro.

```
EDIT1 MSGEDIT ((I,CL3'RAL',CL3'NYC'))
```

Note that only the function, data, and AT operands are coded for this macro; the TO operand must not be coded for an insert operation.

Example 2:
Insertion of a character string after every 50 bytes of message data: The following MSGEDIT macro might be coded in the outbuffer subgroup of a Message Handler assigned to an application program to cause the EBCDIC Z character (specified as a record delimiter by the RECDEL= operand of the TPROCESS macro creating the process entry specified as the destination of the message) to be inserted after every 50 bytes of message data.

```
EDIT2 MSGEDIT ((I,C'Z',(50,EDITOPT)))
```

Note that no TO operand is coded and that only one group is specified. EDITOPT is the name of a halfword option field created by an OPTION macro and initialized with zeros by the OPDATA= operand of the TPROCESS macro creating the process entry specified as the destination of this message.

Example 3:
Replacement of one character string in a message with another character string: The following MSGEDIT macro is coded in the inheader subgroup; it causes the character string BOS to be replaced with the character string OMAHA wherever the former string appears in the first segment of the message (remember, however, that the *entire* character string BOS must occur in the segment in order for MSGEDIT to operate on it). If a buffer is not long enough to accommodate the longer character string, TCAM will dynamically allocate extra units to the buffer as needed. This allocation is automatic.

```
EDIT3 MSGEDIT ((RA,CL5'OMAHA',CL3'BOS'))
```

Note that no TO operand is coded. The A in the function operand specifies that the AT character string is to be deleted and that the O in OMAHA is to be positioned at the location occupied by the B in BOS. If the TO operand had been coded BOS, all data in the segment between the first BOS and a second BOS would be deleted. If the segment contained no second BOS, the remove operation would not take place; the macro would not execute, and control would pass to the next macro.

Example 4:
Insertion and Replacement: A single MSGEDIT macro might be issued in an inheader subgroup to accomplish the two editing functions described above. This macro would cause RAL to be inserted after each NYC in the first segment, and would also cause BOS to be replaced with OMAHA each time the former character string appeared in the first segment.

```
EDIT4 MSGEDIT ((I,CL3'RAL',CL3'NYC'),
               (RA,CL5'OMAHA',CL3'BOS'))
```

Example 5:
Deletion and Contraction: The following MSGEDIT macro might be issued in the inheader or outheader subgroup. It causes the ten bytes immediately following the current position of the scan pointer to be deleted; all data following the deleted ten bytes in the first message segment is shifted to the left ten spaces to fill in the space occupied by the deleted data. The shift may result in an empty unit at the end of this buffer; empty units are dynamically deallocated and returned to the available unit queue when the buffer leaves the MH group.

```
EDIT5 MSGEDIT ((R,,,(10)))
```

Note that the data and AT operands were not coded, since their default values are CONTRACT and SCAN, respectively. Figure 18 illustrates how a single buffer containing an entire message might look before and after this macro was executed. Assume that the units are 64 bytes long, that the buffer consists of two units, and that the second unit contains only six bytes of data before the MSGEDIT macro is executed. Assume also that all of the ten bytes immediately following the position of the scan pointer contain meaningful data (i.e., none of the bytes contain blanks).

Note that after the deletion was made, all data following the deleted characters was moved ten bytes to the left; as a result the second unit contains no meaningful data after the remove operation.

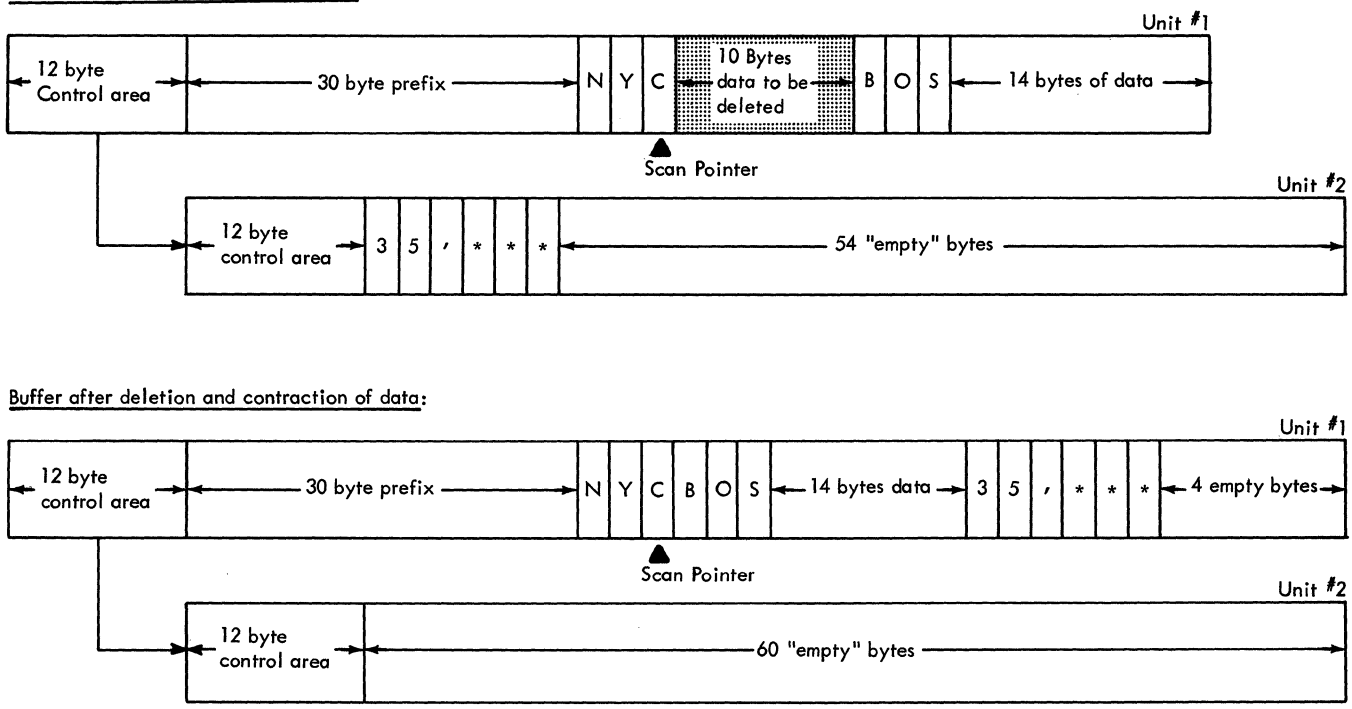


Figure 18. Deletion of Data from a Message Segment, followed by Contraction of the Segment; KEYLEN=60 and BUFSIZE=120.

Example 6:
Insertion of Idle Characters: The following macro, when coded in an inbuffer or outbuffer subgroup, causes 13 EBCDIC idle characters (X'17') to be inserted whenever a period is encountered in a message.

```
EDIT6 MSGEDIT ((I,(X'17',13),CL1'.'))
```

Example 7:
Insertion of a Record Delimiter: The following macro, when coded in an inbuffer or outbuffer subgroup, causes the logical record delimiter X to be substituted for the character D wherever the latter character appears in a message. The X delimiting character, which would be coded in the RECDEL= operand of a TPROCESS macro, is considered by a GET command issued by an application program to be the delimiter of a logical record.

```
EDIT7 MSGEDIT ((RA,DELIMIT,CL1'D'))
```

Example 8:
Miscellaneous Examples: The following MSGEDIT macro, when coded in an inbuffer or outbuffer subgroup, causes the character string OUT and the ten characters immediately following OUT to be removed from a message segment wherever OUT appears in a segment. Data following the 13 deleted characters is moved to the left to fill the gap caused by the deletion. EBCDIC blanks are counted as characters in this example.

```
EDIT8 MSGEDIT ((RA,CONTRACT,CL3'OUT',(10))),BLANK=NO
```

The following MSGEDIT macro, when coded in an inbuffer or outbuffer subgroup causes the data between every R character and E character to be replaced with the character string EPLAC. If the data being deleted occupies less space than the replacement string, the data in the buffer is automatically spread out to make room for the insertion, and another buffer unit is added to the buffer if necessary. If the data being deleted occupies more space than the replacement string, data to the right of the replacement string is automatically moved to the left to fill the gap.

```
EDIT9 MSGEDIT ((R,CL5'EPLAC',CL1'R',CL1'E'))
```

The following MSGEDIT macro, coded in an inbuffer or outbuffer subgroup, causes the characters occupying the tenth through twentieth bytes of each buffer to be deleted, and the remaining data to be shifted left to fill the gap caused by deletion.

```
EDIT10 MSGEDIT ((R,,9,20))
```

The following MSGEDIT macro, coded in an inheader or outheader subgroup, causes the character occupying the byte at which the scan pointer is currently pointing to be removed; subsequent data in the segment is shifted one byte left to fill the gap. Note the defaults.

```
EDIT11 MSGEDIT ((R,,,SCAN))
```

The following MSGEDIT macro, coded in an outbuffer subgroup, causes three EBCDIC SYN control symbols (X'32') to be inserted in each segment, beginning at the thirty-first byte.

```
EDIT12 MSGEDIT  ((I,(X'32',3)31))
```

The following MSGEDIT macro, coded in an inbuffer or outbuffer subgroup causes the EBCDIC blank character (X'40') to be replaced by 13 EBCDIC idle characters (X'17') wherever a blank occurs (BLANK=NO must be specified for this operation). In addition, the character string DOLLARS is replaced with the character $ wherever it appears, and two blanks are inserted after each period in the message.

```
EDIT13 MSGEDIT ((RA,(XL1'17',13),CL1' '),
                (RA,CL1'$',CL7'DOLLARS'),
                (I,CL2' ',CL1'.')),BLANK=NO
```

NOTE: When multiple operations are performed by a single MSGEDIT macro, the data inserted by insert operations is not considered when remove operations are performed.

Thus in the above example, the two blanks inserted after each period would not be replaced by 13 idle characters each.

MSGFORM Macro Instruction

The MSGFORM macro

- Puts line control characters into outgoing messages,
- Permits specification and overriding of blocking factors for outgoing messages,
- Indicates whether an outgoing message is to be transmitted in transparent or non-transparent mode,
- May be specified in the outheader subgroup only.

The MSGFORM macro is optional; if specified, it may be included in an outheader subgroup only. The MSGFORM macro should be coded only in the outheader subgroup of a Message Handler assigned to a line group, and not in the outheader subgroup of the MH for an application program. The MSGFORM macro permits the user to divide his outgoing messages into logical blocks of data. The user specifies blocking factors in the operands of the TERMINAL or MSGFORM macro; the blocking factors specified in MSGFORM override those specified in TERMINAL. If MSGFORM is coded, TCAM inserts appropriate blocking control characters into outgoing messages at the beginning and end of each message and at the locations indicated by the TERMINAL or MSGFORM operands. No buffer space need be reserved for the characters inserted by MSGFORM. MSGFORM inserts EOA, ETX, and EOT characters where needed. These and the blocking characters are not inserted at the time MSGFORM is executed; rather, the characters are inserted after all executable macros in the outgoing group have been executed. For IBM 2260 (Remote), IBM 2265, and BSC stations, STX characters are also inserted. For more information on the line-control scheme utilized by TCAM, see *Defining Terminal and Line Control Areas.*

The MSGFORM macro has the following format:

Name	*Operation*	*Operands*
[symbol]	MSGFORM	[BLOCK=integer] [,SUBBLCK=integer] [,SENDTRP={YES / NO}]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see *symbol* entry in the *Glossary*).

BLOCK=integer

Function: Specifies the number of bytes in each block of data for outgoing messages in transparent or nontransparent mode.
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing X' ' or XLn' ' characters must be specified.
Maximum: 65535 or a hexadecimal field two bytes in length.
Notes: If this operand is not specified, the value used is that specified by the *blocksize* suboperand of the NTBLKSZ= operand of the TERMINAL macro, or by the TBLKSZ= operand of the TERMINAL macro for the destination station.

TCAM inserts an EOB or an ETB line control character after each number of bytes specified by *integer.*

SUBBLCK=integer

Function: Specifies the number of bytes per ITB character for outgoing messages in nontransparent mode to BSC stations.
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing X' ' or XL1' ' characters must be specified.
Maximum: 255 or a hexadecimal field one byte in length.
Notes: If this operand is not specified, the value used is that specified by the *subblocksize* operand of the NTBLKSZ= operand of the TERMINAL macro for the destination station.

TCAM inserts an ITB control character after each number of bytes specified by *integer.*

SENDTRP={YES / NO}

Function: Specifies whether this message is to be sent out in transparent mode.
Default: SENDTRP=NO
Format: YES or NO.
Notes: YES specifies that this message is sent out in transparent mode. SENDTRP=YES should not be coded unless the message is being sent to a BSC station.

NO specifies that the message is sent out in nontransparent mode.

MSGGEN Macro Instruction

The MSGGEN macro:

- Generates an unqueued message,
- Is optional in inmessage and outmessage subgroups,
- May be issued more than once in a subgroup.

MSGGEN generates a message if the errors specified by the error mask operand match the bits set in the message error record (see *Appendix B* for a description of the message error record). If a zero mask is specified, the message is generated unconditionally. The generated message bypasses all normal functions, such as MH processing, queuing, logging, and buffer requesting. The MSGGEN macro informs the user of an error more rapidly than does the ERRORMSG macro, but does not return the header of the message in error, as the latter macro does.

If MSGGEN is specified in an inmessage subgroup, the generated message, as specified by an operand, is sent to the originating station; if specified in an outmessage subgroup, the message is sent to the destination station. MSGGEN may be specified more than once within a subgroup.

Name	*Operation*	*Operand*
[symbol]	MSGGEN	[mask],{message / fieldname} [,CONNECT={AND / OR}] [,CODE={tablename / NO}]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

mask

Function: Specifies the five-byte bit configuration used to test the message error record for the message (see the description of the message error record in *Appendix B*).
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing characters must be specified. If X' ' is used, leading zeros must be coded. If XL5' ' is used, leading zeros may be omitted.
Maximum: 16777215 or a hexadecimal field five bytes in length.
Notes: Omitting the operand, or an all-zero mask, causes unconditional execution.

{message / fieldname}

Function: Specifies the message or the location of the message to be sent to the originating or destination station, depending on whether MSGGEN is issued in an inmessage or outmessage subgroup respectively.
Default: None. This operand must be specified.
Format: message or fieldname. *message* is the actual message to be sent, and has a maximum length of 24 bytes. It must be framed, either by C' ', CLn' ', X' ', or XLn' ' framing characters. *fieldname* is the symbolic name of the field containing the message. It must not be specified with framing characters.
Notes: The message may be specified in EBCDIC and translated as specified by the CODE= operand, or it may be specified in line code if no translation is to occur.

The field referred to by *fieldname* must have as its first byte the hexadecimal count equal to the number of bytes in the rest of the field. The maximum number of bytes in the message portion of the field is 24.

All line control characters, including the EOT, must be coded by the user in his message, with the following exceptions:

- TCAM provides the EOA line-control characters for the IBM 1030, IBM 1050, IBM 1060, IBM 2740, 115A and 83B3 stations.
- TCAM provides an EOT character for BSC stations.

If the user inserts block-checking characters (i.e., EOB) in MSGGEN messages directed to a start-stop station, no checking occurs. For BSC stations, the presence of block-checking characters will cause checks to be made. Messages sent out by MSGGEN are never transmitted in transparent mode.

CONNECT={AND / OR}

Function: Specifies the type of logical connection to be made between the mask and the message error record.
Default: CONNECT=OR
Format: AND or OR.
Notes: AND specifies that the macro is to be executed only if *all* of the bits specified by *mask* are on in the message error record.

OR specifies that the macro is to be executed if *any* bit specified by *mask* is on in the message error record.

CODE={**tablename**}
{**NO** }

Function: Specifies the type of translation for the generated message.
Default: None. Specification optional.
Format: tablename or NO. *tablename* is specified as described for the TRANS= operand of the line group DCB macro. Register notation may not be used. The user may devise and specify his own translation table as described for the CODE macro.

NO specifies that the message is not to be translated. If this operand is omitted, the message is translated using the translation table specified in the line group DCB for the line. If this operand is omitted and no translation table is specified in the line group DCB macro, no translation occurs.

A message generated by MSGGEN may not be directed to a distribution list or to an application program when specified in an inmessage subgroup.
Note: A premature disconnection on a switched line will prevent the message from being returned to the originating station; the message is lost.

MSGLIMIT Macro Instruction

The MSGLIMIT macro:

- Limits the number of messages to or from a station during a single transmission sequence,
- Is effective only when used with a nonswitched line,
- Is optional in the inheader and outheader subgroups of an MH.

MSGLIMIT limits the number of messages that can be transmitted to or accepted from a single station on a nonswitched line following a positive response to invitation or selection. If coded in an inheader subgroup, MSGLIMIT limits the number of messages entered by a station or application program during a single transmission sequence; if coded in an outheader subgroup, MSGLIMIT limits the number of messages sent to a station or application program during a single transmission sequence. For instance, for stations that are polled, MSGLIMIT in the inheader subgroup causes the current station to cease to be polled once the specified maximum number of messages is reached; the next entry is then polled. If no limit is set for polled stations, each station is polled until it has no more messages to enter (negative response).

MSGLIMIT has no effect when used with a switched line. The MSGLIMIT macro is optional in inheader and outheader subgroups. Its use is suggested for IBM 2260 and 2265 terminals; the outheader subgroup for these terminals should include a MSGLIMIT macro specifying a limit of one message in inquiry applications (in order to ensure that a response message is not erased before it can be read). For a description of the use of MSGLIMIT with a contention terminal, see *Transmission Priority for Nonswitched Contention Stations* in the chapter *Terminal and Line Control Area Definition.*

NOTE: If a MSGTYPE macro or user code is used to cause MSGLIMIT to be executed only for certain types of messages, only those subsequent messages examined by the same MSGLIMIT macro will be counted when the limit for a station is being determined.

Name	*Operation*	*Operand*
[symbol]	MSGLIMIT	{integer} {opfield}

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

{integer}
{opfield}

Function: Specifies the number of messages or the location of the number of messages that the user wishes to transmit to or receive from each terminal on the line.
Default: None. This operand must be specified.
Format: integer or opfield. *integer* may be specified either in decimal or hexadecimal format. If hexadecimal format is specified, framing X' ' or XLn' ' characters must be coded. *opfield* must be the same as the name of a one-byte option field defined by an OPTION macro.
Maximum: For *integer*, 255 or a hexadecimal field one byte in length.
Notes: If *integer* is specified, all stations processed by this MH are limited to the same MSGLIMIT value.

If *opfield* is specified, the option field contains the limit of consecutive message transmissions that is allowed to or from a station. Use of this operand allows the message limit specification to be different for each station. If the option field cannot be found, MSGLIMIT does not execute and a return code of X'04' is set in the low-order byte of register 15.

MSGTYPE Macro Instruction

The MSGTYPE macro:

- Controls the path of a header through an MH,
- Is optional in inheader and outheader subgroups (and not permitted in any other subgroup),
- May be used more than once in a subgroup.

MSGTYPE enables the user to categorize incoming or outgoing messages into two or more message types, each of which he processes in a different manner. The next nonblank character or character string in a header buffer (after the current setting of the scan pointer) is compared with a character or character string specified by the operand of MSGTYPE. If the two characters or character strings are identical, the instructions between this MSGTYPE macro and the next MSGTYPE or (if this is the last MSGTYPE macro in the subgroup) the next delimiter macro are executed. If the two characters or character strings are not identical, those instructions are not executed (the scan pointer is reset to its position prior to the comparison). Instructions between a MSGTYPE macro with no operand and the next delimiter are executed for all message headers whose character string has not matched the operand specified in a previous MSGTYPE macro. (The MSGTYPE macro with no operand should be the last MSGTYPE macro issued in the subgroup.) These instructions are bypassed if the message was previously handled by a MSGTYPE macro with a message-type character operand.

Use of MSGTYPE is optional. Any number of MSGTYPE macros may be issued within a subgroup, provided that they all examine the same position in the buffer for the message-type characters. Only one field in a header per inheader or outheader subgroup may contain message-type characters, and only one sequence of code beginning with a MSGTYPE macro is executed in an inheader or outheader subgroup for any one incoming or outgoing message. MSGTYPE may be used only within inheader and outheader subgroups.

The use of MSGTYPE is discussed in the *Variable Processing within a Message Handler* section of this chapter.

Name	*Operation*	*Operand*
[symbol]	MSGTYPE	[conchars[,BLANK={YES | NO | char}]]

symbol

Function: Name of macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

conchars

Function: Specifies the character or character string to be compared with the message type field in the message header.
Default: None. Specification optional.
Format: One to eight nonblank character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters.
Notes: If the *conchars* field matches the field found in the message, all macro instructions between this MSGTYPE macro and the next MSGTYPE macro (or the next delimiter, if there is not another MSGTYPE) are executed. If the control characters do not match the header field, the MSGTYPE macro does not execute and control passes to the next MSGTYPE macro in the subgroup or, if this was the last MSGTYPE macro in the subgroup, to the next delimiter macro.

If this operand is omitted, the group of macro instructions that immediately follows this MSGTYPE will process all message headers not handled by a preceding MSGTYPE macro with a nonblank operand. A MSGTYPE macro with no *conchars* operand may be used only as the last of a series of MSGTYPE macros (with nonblank operands).

If MSGTYPE macros are used both with and without the *conchars* operand, either some message type field should always be specified, or care should be taken, if the field is omitted, that the next field cannot match any of the strings specified by the *conchars* operand in the series of MSGTYPE macros.

BLANK={YES / NO / char}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES
Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: This operand is meaningless unless the *conchars* operand is also specified. YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored, and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars.*

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

Example:
The beginning of an MH using MSGTYPE is shown in Figure 19. Type A messages are processed and forwarded to terminal NYC, type B to terminal BIX, and all others to an application program.

Name	*Operation*	*Operands*	*Comments*
MHA	STARTMH	LC=OUT	LC= must be coded for STARTMH
	INHDR		Delimiter
	SEQUENCE ORIGIN DATETIME		Macro instructions executed for all header segments
	COUNTER	FIELD	Count incoming segments
	MSGTYPE	C'A'	Test for Type A messages
	— —		Macro instructions executed for all Type A messages
	FORWARD	DEST=CL3'NYC'	
	MSGTYPE	C'B'	Test for Type B messages
	— —		Macro instructions executed for all Type B messages
	FORWARD MSGTYPE	DEST=CL3'BIX'	
	FORWARD	DEST=CL8'PROCESSQ'	Macro instructions executed for all other message types
	INMSG etc.		Delimiter

Figure 19. Example of Using the MSGTYPE Macro Instruction

ORIGIN Macro Instruction

The ORIGIN macro:

- Checks the validity of the origin field in a message header,
- Sets a bit in the message error record for the message if the origin field is invalid,
- Permits identification of a switched station calling the computer,
- Is optional in the inheader subgroup and is not permitted in any other subgroup.

The function of the ORIGIN macro depends upon the kind of connection made with the station. For nonswitched stations, ORIGIN verifies that the origin field in the header contains the symbolic name of the station that was invited to send the message (that is, the origin field is compared with the name of the terminal table entry for the station that was contacted). If the names are not the same, an error flag is set in bit 1 of the message error record for the message.

For switched stations, ORIGIN both checks the validity of the origin field in the header and identifies the calling station to TCAM. Unless the calling station is a BSC station that transmits a unique ID sequence upon successfully calling the computer, TCAM does not know what station is on the line until an ORIGIN macro is issued in the MH. Once an ORIGIN macro is issued, TCAM compares the name in the origin field of the message header with the terminal table entries for the stations assigned to lines in the line group to which the line connecting the station to the computer is assigned. If a match is found, TCAM assumes that the station named in the origin field is the calling station. If no match is found, an error flag is set in bit 1 of the message error record for the message.

Inheader subgroups for switched lines to stations that do not have unique ID sequences and that may call the computer and enter messages should include an ORIGIN macro, as this is the only means TCAM has of identifying the calling station in this situation.

An inheader subgroup that handles only messages entered by BSC stations having unique ID sequence requires no ORIGIN macro. When an ORIGIN macro is included in the inheader subgroup that processes header segments entered by such a station, the name in the origin field, if valid, takes precedence over the name associated with the ID sequence in the invitation list; that is, TCAM assumes that the station named in the origin field is the station that entered the message.

For switched stations assigned to a line for which a TERMINAL macro coded UTERM=YES has been issued, the position of ORIGIN in the inheader subgroups determines whether the option fields assigned to the line or those assigned to the station will be updated by MH macros when a station calls the computer. Inheader macros executed prior to ORIGIN refer to option fields assigned to the line by a TERMINAL macro coded for the line, while macros executed after ORIGIN refer to option fields assigned to the station by a TERMINAL macro coded for that station. (For a more detailed discussion of the relationship between the ORIGIN macro and the TERMINAL macro coded for a line, see *Coding the Terminal Macro for a Line* in *Defining Terminal and Line Control Areas.*)

If ORIGIN is used with a message having a multiple-buffer header and entered from a station on a switched line, ORIGIN must be executed for the first header buffer in order to effectively identify the station.

A CODE macro must be issued prior to ORIGIN (unless the line code is EBCDIC).

NOTE: Care must be taken in entering a character string in an origin field in the message header to ensure that it matches a terminal-table entry. A character string entered in lower-case characters from an IBM 2770 station, for example, will not match a terminal-table entry name that is in uppercase characters.

The ORIGIN macro has the following format:

Name	*Operation*	*Operand*
[symbol]	ORIGIN	[integer / X'FF']

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

[integer / X'FF']

Function: Specifies the number of characters in the origin field of a message header.
Default: X'FF'
Format: Decimal or hexadecimal. If hexadecimal format is specified, framing X' ' or XL1' ' characters must be used.
Maximum: 8
Notes: If *integer* is specified, that many characters are accessed and considered to be the origin field. Embedded blanks are ignored.

X'FF' indicates that the origin field is of variable length. The origin field is considered to end at the next blank.

NOTE: If an ORIGIN macro determines that the source of a message on a nonswitched line is invalid, a return code of X'04' is set.

PATH Macro Instruction

The PATH macro:

- Alters a path-switch byte, thereby permitting dynamic variation of the path of a message through an MH,
- Is optional in inheader, inbuffer, outheader, and outbuffer subgroups (and permitted in no other subgroup).

One-byte option fields are used to maintain switches known as *path switches.* These switches are located in option fields defined by OPTION macros. The switches must be set initially by the OPDATA= operand of the TERMINAL or TPROCESS macro (if the option fields are not initialized, the PATH macro provides a return code of X'00'). The setting of path switches is examined by each delimiter macro as the message reaches the subgroup it controls (the user specifies by the PATH operand of each delimiter which path-switch bytes are to be examined). More than one option field may be specified for each station; each path-switch byte so defined consists of eight binary switches.

If any of the binary switches tested by a delimiter is on, the subgroup controlled by that delimiter is executed; if none of the binary switches tested is on, control passes to the next delimiter.

The user may specify a character string (consisting of one to eight nonblank characters). If this character string appears in the header of a message, the PATH macro having character string sets one or more specified path switches. If no character string is specified, the switches are set unconditionally.

PATH may specify any number between zero and 255 inclusive. The switches remain set until reset by a PATH macro specifying the same option field, until modified by user code and LOCOPT, or until modified by a DATOPFLD operator command.

The use of PATH is discussed in the *Variable Processing within a Message Handler* section of this chapter.

Name	*Operation*	*Operand*
[symbol]	PATH	switch,opfield [,conchars[,BLANK={YES / NO / char}]]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

switch

Function: Specifies the path switch setting to be made for the byte residing in the option field named by the *opfield* operand.
Default: None. This operand must be specified.
Format: Decimal or hexadecimal. If hexadecimal format is specified, framing X' ' or XL1' ' characters must be used.
Maximum: 255 or a hexadecimal field one byte in length.
Notes: If 0 is specified, all eight path switches are turned off. If 255 (or X'FF') is specified, all switches are turned on.

opfield

Function: Specifies the path-switch byte to be operated upon.
Default: None. This operand must be specified.
Format: The name of a one-byte field in the option table as defined by an OPTION macro.
Notes: If the option field cannot be found, the path-switch byte is not operated upon and a return code of X'00' is set in the low-order byte of register 15.

If PATH is coded in the incoming group of an MH for a line group, the specified option field for the station entering the message is operated upon. If PATH is coded in the outgoing group of a line MH, the specified option field for the destination station is

operated upon. If PATH is coded in the outgoing group of an MH assigned to an application program, the option field associated with the process queue to which the GET macro is directed is operated upon. If the macro is coded in the incoming group of an MH assigned to an application program, the option field for the process entry associated with the DCB named in the PUT macro is operated upon.

conchars

Function: Specifies the character or character string that, if found in the header as the next nonblank field, causes execution of the function.
Default: None. Specification optional.
Format: One to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters.
Notes: This operand should be coded only in PATH macros issued in an inheader or outheader subgroup.

If this operand is omitted, the PATH function is performed unconditionally. If the next field in the header does not match this operand, the function is not performed.

BLANK={YES / NO / char}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES
Format: YES or NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: This operand is meaningless unless the *conchars* operand is also specified. YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars.*

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

Example:
Figure 20 shows the outline of an inmessage group of an MH. Messages with A, B, or C in an appropriate field are routed through the incoming group by PATH macro instructions. The switch settings enable the user to select appropriate inbuffer and inmessage subgroups. Message type A passes through the first inbuffer subgroup and the second inmessage subgroup, etc.

Warning:
In the case of multiple-buffer headers, the entire control-character field must appear in the first header segment.

Name	*Operation*	*Operands*	*Comments*
VARYPATH	STARTMH INHDR	LC=OUT	Inheader subgroup executed for all messages.
	• • •		
	PATH	4,SWITCH,C'A'	Sets switch for Type A messages (not executed for others).
	• • •		
	PATH	2,SWITCH,C'B'	Sets switch for Type B messages.
	PATH	1,SWITCH,C'C'	Sets switch for Type C messages.
	• • •		
	INBUF	4,SWITCH	This inbuffer subgroup is executed if switch is 4.
	• • •		
	INBUF	2,SWITCH	This inbuffer subgroup is executed if switch is 2.
	• • •		
	INBUF	1,SWITCH	This inbuffer subgroup is executed if switch is 1.
	• • •		
	INMSG	3,SWITCH	This inmessage subgroup is executed if switch is 1 or 2.
	• • •		
	INMSG	4,SWITCH	This inmessage subgroup is executed if switch is 4.
	• • •		
	INEND		

Figure 20. Example of Using the PATH Macro Instruction to Vary MH Processing

PRIORITY Macro Instruction

The PRIORITY macro:

- Specifies priority handling for messages,
- Is optional in an inheader subgroup of the MH,
- May be used more than once in the subgroup.

PRIORITY provides handling of outgoing messages according to priority levels. The priority level may be entered in the message header by the user, or it may be specified by an operand of the PRIORITY macro. The permissible priority levels for each station or application program are specified in the TERMINAL or TPROCESS macro for that destination. If queuing by line rather than queuing by terminal is specified, the first TERMINAL macro for the line contains the permissible priority levels for all stations on the line; if subsequent TERMINAL macros for the same line specify priority levels, they are ignored. If a message priority is requested that is not permitted, the message is assumed to have the next lower permissible priority. The PRIORITY macro must be specified within the subgroup in the same relative order as the header field on which it acts.

Absence of the PRIORITY macro causes a priority level of zero to be assigned to the messages.

For more information on message priority, see *Message Priority* in *Terminal and Line Control Area Definition.*

NOTE: TCAM converts the decimal priority levels specified by the LEVEL= operand of the TERMINAL or TPROCESS macro to their one-byte hexadecimal equivalents. If the priority is specified in a message header, it may occupy a one-byte field and should provide the hexadecimal equivalent of a decimal priority level specified by the LEVEL= operand of the TERMINAL or TPROCESS macro. For example, if PRIORITY is executed after a CODE macro (i.e., the message segment has been translated from line code to EBCDIC), and if messages entered by a particular station may be assigned priorities of 1, 2, A, B, or C on output, the LEVEL= operand of the TERMINAL macro for that station should be coded LEVEL= (193, 194, 195, 241, 242). Here, 193 is the decimal representation of the hexadecimal equivalent of the EBCDIC character A; 241 is the decimal representation of the hexadecimal equivalent of the EBCDIC character 1, etc. In this case, a message assigned a line-code character 1 as its priority would be higher in priority than a message assigned a line-code character A, B, or C.

On the other hand, if PRIORITY is executed prior to a CODE macro, and if the messages are being entered by a 1050 station and may be sent with priorities of 1, 2, A, B, or C, the LEVEL= operand of the TERMINAL macro should be coded LEVEL= (2, 4, 226, 228, 231); here 2 is the decimal representation of the hexadecimal equivalent of the 1050 line-code character 1; 226 is the decimal representation of the hexadecimal equivalent of the 1050 line-code character A, etc. In this case, a message assigned a line-code character A as its priority would be higher in priority than a message assigned a line-code character 1 or 2.

Name	*Operation*	*Operands*
[symbol]	PRIORITY	[integer] [,conchars] [,BLANK=<u>YES</u> | NO | char]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

integer

Function: Specifies the priority level to be assigned to this message.
Default: None. Specification optional.
Format: Unframed decimal integer.
Maximum: 255

Notes: If this operand is omitted, TCAM assumes that the priority level is contained in the next nonblank byte following the current setting of the scan pointer. If the priority level is not one that the TERMINAL or TPROCESS macro specifies as permissible, the next lower permissible priority is assumed.

conchars

Function: Specifies the character or character string that, if included in the message header, causes execution of the PRIORITY macro specifying that string.
Default: None. Specification optional.
Format: One to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X'' or XLn' ' characters.
Notes: If this operand is omitted, PRIORITY is specified unconditionally. If the control characters do not match, the PRIORITY macro does not execute and control passes to the next instruction.

If this operand is specified, but the *integer* operand is omitted:

- The message priority is assumed to be contained in the message header as the next nonblank character following control characters.
- A comma must precede the *conchars* operand.

BLANK={**YES** / **NO** / **char**}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES
Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: This operand is meaningless unless the *conchars* operand is also specified.

YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars.*

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

NOTE 1: If the *integer* and *conchars* operands are omitted, the priority is assumed to be in the message header in the next nonblank character following the current setting of the scan pointer.

NOTE 2: In the case of multiple-buffer headers, the priority, if desired, must be determined for the first header segment to pass through the inheader subgroup. This can be ensured in one of two ways:

1. The priority field in the header, if used, must be in the first header segment (and for messages from buffered terminals, in the first hardware buffer if the hardware buffer is smaller than the MCP buffers), or
2. The *integer* operand must be specified to provide the priority, and any control characters used to execute the PRIORITY macro must be in the first buffer (and for messages from buffered terminals, in the first hardware buffer, if the hardware buffer is smaller than the MCP buffers).

Example:
The following examples show the various ways message priority may be specified. It is assumed that the LEVEL= operand of the TERMINAL macro for the destination station is coded LEVEL= (241, 243, 245, 247).

PRIORITY Macro	*Header Fields in EBCDIC*	*Priority given message (decimal notation)*
PRIORITY	5	5
PRIORITY	6	5
PRIORITY 241	3	5
PRIORITY 241, PRI	PRI	241
PRIORITY , PRI	PRI3	243
PRIORITY , PRI	PRO	Priority of 0 is assigned (the macro is not executed)

Figure 21. Example of Using the PRIORITY Macro Instruction

REDIRECT Macro Instruction

The REDIRECT macro:

- Queues a message for an additional destination,
- Is optional in an inmessage or outmessage subgroup of an MH,
- May be specified more than once within a subgroup.

REDIRECT queues a message for a destination in addition to the destinations specified by the FORWARD macro, when errors specified by the mask operand are detected. The bits specified by the error mask operand are compared with the setting of the bits in the message error record for the message. If specified bits in the message error record are on, the REDIRECT macro is executed; otherwise, the REDIRECT macro is not executed.

The additional destination specified may be any single, group, process, or cascade list entry in the terminal table. A distribution list cannot be specified as the additional destination.

REDIRECT may be used to send unsent messages to an application program, or to return them to the originating station, or to send them to the alternate destination when the intended destination is inoperative.

If REDIRECT is specified, it must appear in an inmessage or outmessage subgroup of an MH.

Name	*Operation*	*Operands*
[symbol]	REDIRECT	[mask] [,CONNECT={AND | OR}] [,DEST={destname | opfield | ORIGIN}]

symbol

Function: Name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

mask

Function: Specifies the five-byte bit configuration used to test the message error record for the message (the message error record is described in *Appendix B*).
Default: None. Specification optional.
Format: Decimal or hexadecimal. If hexadecimal format is used, framing characters must be specified. If X' ' is used, leading zeros must be coded. If XL5' ' is used, leading zeros may be omitted.
Maximum: 16777215 or a hexadecimal field five bytes in length.
Notes: Omitting the operand, or an all-zero mask, causes unconditional execution.

CONNECT={AND | OR}

Function: Specifies the type of logical connection to be made between the mask and the message error record.
Default: CONNECT=OR
Format: AND or OR.
Notes: AND specifies that the macro is to be executed only if *all* of the bits specified by *mask* are on in the message error record.

OR specifies that the macro is to be executed if *any* bit specified by *mask* is on in the message error record.

DEST={destname | opfield | ORIGIN}

Function: Specifies the additional destination.
Default: DEST=ORIGIN
Format: destname, opfield, or ORIGIN. *destname* may be up to eight bytes in length and is the name of any single, group, or cascade list entry in the terminal table. It must be specified within framing C' ', CLn' ', X' ' or XLn' ' characters. *opfield* is the unframed name of an option field defined by an OPTION macro, and cannot be named ORIGIN.
Notes: If an invalid destination is specified, REDIRECT does not execute.

opfield refers to an option field up to eight bytes in length, containing the name of the destination. The additional destination is the station

- Specified in the option field assigned to the originating station or application program, if REDIRECT is used in an inmessage subgroup and if the originating station is not a switched station that called the computer to enter the message but did not identify itself by means of a unique ID sequence or by a valid origin field checked by an ORIGIN macro,
- Specified in the option field assigned to this line by a TERMINAL macro coded for a line if the originating station is a switched station that called the computer in order to enter this message but did not uniquely identify itself,
- Specified in the option field assigned to the destination station or application program if REDIRECT is used in an outmessage subgroup.

ORIGIN specifies that the message in error is to be sent to the station from which it originated (in addition to the destinations specified in the message header).

If this operand is omitted or if DEST=ORIGIN is specified, and the originating station is a switched station which has called the computer in order to enter the message, the station must identify itself by means of a unique ID sequence or by means of a valid origin field as checked by an ORIGIN macro. Otherwise TCAM is unable to identify the station of origin and cannot route the message to it.

The SCREEN macro:

- Modifies the Write operation for terminals with display screens,
- Is optional in outheader subgroups of Message Handlers for terminals with display screens.

SCREEN may be used in an outheader subgroup of an MH to specify the type of modification to be made to the Write operation for the 2260 or the 2265. If the user specifies the Write Display Control (WDC) operation, write operations begin at the position of the display cursor. Alternatively, the user may specify the Write Erase (WRE) function so the screen is erased before the next segment is displayed and writing begins at the top of the screen. Or, the user may specify the line on which he wishes to write, using the Write-at-Line-Address (WLA) function.

If the WLA function is used, the user must specify the line address character desired as the first character of the header of the message to be written. If line-control is left in the message, the line-address character should follow any initial line-control characters. The user may insert the line address in the message header by:

1. Including the necessary assembly language instructions in an MH or in an application program.
2. Using the MSGEDIT macro.

The table below gives the appropriate line addresses in EBCDIC for 2260 terminals. The following MSGEDIT macro, executed in an outheader subgroup on the first segment of a message, would place the line address for line number ten in the first byte of the header (assuming that the header contains no line control):

```
LINEADDR   MSGEDIT        ((I,XL1'F9',0))
```

The type of operation (WRE, WLA, WDC) to be performed for a display terminal is specified initially by the user in the TERMINAL macro for that terminal. (See the description of the ADDR= operand of the TERMINAL macro.) When the TERMINAL macro is executed, TCAM sets a byte in the terminal-table entry to indicate the type of Write operation to be performed for all messages directed to this terminal. The SCREEN macro changes the type of operation to be performed by modifying this byte (e.g., from WLA to WRE). However, for remote operations, the change specified by SCREEN does not take effect until the *next* message is sent to the terminal; if WLA was changed by SCREEN to WDC, all messages sent to the terminal after the current message would have a WDC operation performed for them, but the *current* message (i.e., the message being processed when SCREEN executes) would have a WLA operation performed for it. For local operations, the change specified by SCREEN takes effect for the current operation.

When WLA is changed to another Write operation by SCREEN, the user must still place the line address in the header of the current message, since a WLA operation will be performed for this message. If the operation type operand of SCREEN is omitted, then when SCREEN finishes executing, a return code is placed in register 15 to indicate what the setting of the terminal-table byte was before SCREEN changed it. User code may test the return code; if the code indicates that a WLA operation was being performed, a MSGEDIT macro may be executed to insert the line address. Otherwise, the user code may branch around the MSGEDIT macro (see the example following the macro description).

Line Address Characters for the IBM 2260 Terminal

Hexadecimal representation of EBCDIC line address	Selected line
F0	1
F1	2
F2	3
F3	4
F4	5
F5	6
F6	7
F7	8
F8	9
F9	10
FA	11
FB	12

The SCREEN macro has the following format:

Name	*Operation*	*Operands*
[symbol]	SCREEN	{WRE / WLA / WDC} [,conchars[,BLANK={YES / NO / char}]]

symbol

Function: Name of the macro.
Default: None. Specification option.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

{WRE / WLA / WDC}

Function: Specifies the type of write operation for the IBM 2265 (Remote) or the IBM 2260 (Local or Remote).
Default: None. Specification optional.
Format: WRE, WLA, or WDC.
Notes: WRE specifies a Write Erase operation, the erasure of the screen before the next segment is displayed.

WLA specifies a Write at Line Address operation. The user must insert a line address character as the first character (following any initial line-control characters, if line control was left in the message) of the message to be written out.

WDC specifies a Write Display Control operation.

conchars

Function: Specifies the character or character string that, if found in the header as the next nonblank field, causes execution of the function.
Default: None. Specification optional.
Format: One to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters.
Notes: If this operand is omitted, the SCREEN function is performed unconditionally. If the next field in the header does not match this operand, the function is not performed.

A SCREEN macro specifying no WRE, WLA or WDC operation may be issued to check the type of write operation in effect for the message being processed without changing the operation type. See the example below.

If WLA is changed to another Write operation, the user code in the outheader subgroup needs to know whether to cause a line-address character to be inserted in the current message. The following return codes may be returned by SCREEN in register 15:

Code	Operation
X'A0'	WDC
X'B0'	WLA
X'E0'	WRE

BLANK={YES / NO / char}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES
Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X'' or XL1' ' characters.

Notes: This operand is meaningless unless the *conchars* operand is also specified.

YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars*.

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

Example:
The following code might be issued in an outheader subgroup to determine whether the Write-at-Line-Address (WLA) operation is specified in the appropriate byte in the terminal-table entry for the destination station for the message being processed, and to cause a line address of 10 to be placed in the first byte of the message header of the current message if WLA is specified in the terminal-table entry. SCREEN is executed only for type A messages (as determined by a field in the message header and the *conchars* operand of SCREEN).

Name	*Operation*	*Operand*
	OUTHDR	
	SCREEN	,A
	LA	5,X'B0'
	CLR	5,15
	BNE	CONTINUE
	MSGEDIT	((I,XL1'F9',0))
CONTINUE	EQU	*
	next instruction	

Figure 22. Example of Inserting Line Address.

SEQUENCE Macro Instruction

The SEQUENCE macro:

- Checks the input sequence number of an incoming message,
- Inserts the output sequence number in an outgoing message,
- Is optional in inheader and outheader subgroups on non-audio lines (and may not be used in any other subgroup),
- Should be specified only once in each subgroup.

If specified in an inheader subgroup, SEQUENCE scans the input sequence number field in the header of each message. If the sequence number is not one greater than the sequence number of the last message received from the same station or application program, an error flag is set in bit 3 or 4 of the message error record assigned to the message (depending on whether the number is high or low, respectively), and a return code indicating an error is set in register 15.

The header field for the input sequence number may contain up to four characters of sequence (leading zeros may be omitted from the input sequence number entered at the station). This field should contain a decimal representation of the input sequence number, and should be delimited by a blank. For example, if the sequence number is twelve,

the field should consist of a character 1 followed by a character 2 followed by a delimiting blank. At the time SEQUENCE looks at the field, the characters should have been translated into EBCDIC by a CODE macro. The user should reserve five bytes in his header (via the RESERVE= operand of the line group DCB macro) for insertion of the output sequence number, if used.

TCAM maintains internal counters in the terminal table entry to keep track of the incoming and outgoing sequence numbers for each station and application program. If the SEQUENCE macro is issued in an inheader subgroup, the first message from a station or application program must contain the same input sequence number as the input counter for that station or application program. TCAM initially sets each input counter to 1. The next incoming message after 9999 must be numbered 1. Processing continues after the maximum number is reached.

In general, SEQUENCE in an inheader subgroup causes the input counter to be incremented for each message having a correct input sequence number in the header. If, however, a CANCELMG macro causes a message to be canceled, or if a STARTMH macro causes retransmission of a message header segment, the input sequence number is not incremented. In the latter case, the number is incremented only when the segment is successfully retransmitted.

If specified in an outheader subgroup, SEQUENCE places an output sequence number in the header of each outgoing message handled by the MH. The five-byte output sequence number (a blank followed by four EBCDIC characters) is inserted immediately following the byte to which the scan pointer is pointing when SEQUENCE executes. When the first message is sent to a station or application program, a 1 is placed in the output counter for that station or application program; as each succeeding output message is handled, this sequence number is incremented by 1 and the resulting number inserted in the header. (A count is maintained for each station and for each terminal group where group addressing is used.) A message in error routed via a REDIRECT macro retains the output sequence number placed in it.

Use of SEQUENCE is optional. If used, it must appear within an inheader or outheader subgroup. Its position must correspond to the relative position, within the header, of the sequence-number field.

TCAM increments the input sequence number counter in the terminal table entry only if a SEQUENCE macro is issued in the inheader subgroup. TCAM increments the output sequence number counter in the terminal-table entry, automatically, each time that a message is sent to the station or application program.

If SEQUENCE is included in an inheader subgroup handling header segments entered by a switched station that calls the computer to enter the segments, and if the station does not have a unique ID sequence assigned to it, SEQUENCE should not be executed until after an ORIGIN macro has been executed. In this case, ORIGIN is needed to identify the calling station so that SEQUENCE can access the correct input counter.

Name	*Operation*	*Operand*
[symbol]	SEQUENCE	(no operands)

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

There are no operands. Five spaces must be reserved for insertion of an output sequence number; space is reserved by a RESERVE= operand of the line group DCB macro. If insufficient space is reserved, a SEQUENCE macro coded in an outheader subgroup is not executed and a return code of X'04' is set in register 15. For incoming messages, if the input sequence number in the message header is low, a X'04' return code is set, and if the number is high or if there is no valid decimal number in the header field being examined, X'08' is set. If the source of an incoming message is not known at the

time SEQUENCE is reached in an inheader subgroup, the macro does not execute and a return code of X'0C' is placed in register 15. In none of these cases is the input sequence counter in the appropriate terminal-table entry incremented.

Note: Continuity of sequence numbers is maintained by the continuation and warm start capabilities of the TCAM restart facility.

SETEOF Macro Instruction

The SETEOF macro:

- Sets a bit in the buffer prefix to indicate an EOF message,
- Is optional in the outheader subgroup of the MH assigned to an application program (and should be coded in no other).

The SETEOF macro is used to identify the last message in a data file being processed by an application program. When the application program receives a message flagged by SETEOF, the next GET or READ/CHECK it issues after the complete message has been received will cause control to be passed to the routine whose address is specified by the EODAD= operand of the application program input DCB for the destination queue accessed by the GET or READ. Thus, by issuing a SETEOF macro, the user causes the application program to stop obtaining work units from one or more destination queues and do whatever is specified by the routine located at the EODAD address.

The SETEOF macro is issued in the outheader subgroup of the outgoing group of the MH handling messages routed to an application program.

Note: In the case of multiple-buffer headers, SETEOF must be executed for the first header buffer to be effective.

The SETEOF macro has the following format:

Name	*Operation*	*Operand*
[symbol]	SETEOF	[conchars[,BLANK={YES / NO / char}]]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

conchars

Function: Specifies the character or character string that, if found in the header as the next nonblank field, causes execution of the function.
Default: None. Specification optional.
Format: One to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters.
Notes: If this operand is omitted, the SETEOF function is performed unconditionally. If the next field in the header does not match this operand, the function is not performed.

BLANK={YES / NO / char}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES

Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: This operand is meaningless unless the *conchars* operand is also specified.

YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars.*

chars specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

SETSCAN Macro Instruction

The SETSCAN macro:

- Explicitly moves the scan pointer forward or backward, or
- Returns in a designated register the address of the last character of a specified character string, or
- Returns in register 15 the current address of the scan pointer.
- Is optional in inheader and outheader subgroups (and not permitted in any other subgroup),
- May be specified more than once in the same subgroup.

The SETSCAN macro explicitly repositions the scan pointer forward or backward in the buffer, if specified. After the previous macro has executed, the scan pointer is positioned at the last character of the field acted upon by that macro. SETSCAN moves the scan pointer a specified number of nonblank characters, to a specified character, or to the last character of a specified character string, so that one or more fields is skipped. The scan pointer is left at its new position.

Alternatively, the scan pointer is not moved. Instead, either a designated character string is located, and the address of the last character of the string is placed in a specified register, or the current address of the scan pointer is placed in register 15.

Use of SETSCAN is optional in inheader and outheader subgroups, where it may be used as many times as desired.

NOTE 1: When an outgoing message is processed by an outheader subgroup, STARTMH positions the scan pointer to the last reserve character in the buffer or, if there are no reserve characters, to the last byte of the buffer prefix or, if there is a machine EOA sequence, to the last byte of the EOA. If this is not the location of the next header field to be processed, the user may employ SETSCAN to move the scan pointer to the byte immediately preceding the next header field to be processed.

NOTE 2: For an inheader subgroup that handles messages from stations designated as operator control terminals, if LC=IN is coded in the STARTMH macro, a SETSCAN macro should be coded as the first functional macro of the subgroup, with a CODE macro being the second functional macro (CODE tests for operator commands). The SETSCAN macro should move the scan pointer past the line control characters (not including the EOA), leaving it pointing to the byte immediately preceding the first byte of meaningful header data.

The SETSCAN macro has the following format:

Name	*Operation*	*Operands*
[symbol]	SETSCAN	{skipchars / integer} [,BLANK={YES / NO / char}] [,POINT={BACK / FORWARD}] [,MOVE={RETURN / KEEP}] [,RESULT={(register) / (15)}]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

{skipchars / integer}

Function: Specifies the new location of the scan pointer, either a character string to be located or a number of characters to be advanced.
Default: None. This operand must be specified.
Format: skipchars or integer. *skipchars* can be one to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used, the string must be framed with X' ' or XLn' ' characters. *integer* is a decimal integer in decimal format.
Maximum: For *integer*, 65535.
Notes: *skipchars* is the character or character string to which the scan pointer is to be moved when MOVE=KEEP is specified, or whose address is to be placed in the register specified by RESULT=(register) when MOVE=RETURN is specified. If MOVE=KEEP is specified, SETSCAN operates *across buffers* until the string is found. If MOVE=RETURN is specified, the character string must be located in the current buffer; otherwise, SETSCAN does not execute and a return code is set. If register 15 is specified to contain the address of the string and the string is not found in the buffer, a return code of X'00' is set in the low-order byte of register 15. If another register is specified for the address of the string and the string is not found in this buffer, a return code of X'04' is set in the last byte of register 15 (see the description of MOVE=RETURN below). The pointer should not be moved to a position outside the header. If *skipchars* is specified, POINT=BACK may *not* be specified.

integer is the number of nonblank characters to be skipped. If 0 is specified in place of *integer*, TCAM returns the main-storage address to which the scan pointer is currently pointing. If an attempt is made to set the scan pointer outside of the current buffer, SETSCAN does not execute and control passes to the next macro. If *integer* is coded MOVE=KEEP is assumed by TCAM, even if MOVE=RETURN is coded in the macro (unless 0 is coded in place of *integer*). If 0 is coded, the MOVE= and RESULT= operands are ignored.

BLANK={YES / NO / char}

Function: Specifies whether EBCDIC blank characters are to be ignored when being counted or when encountered in the character string in the message header being compared to the string specified by the *skipchars* operand, or whether blanks are to be counted as characters or as part of the header string when encountered in it. If EBCDIC blanks are to be counted as characters or as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when skipping or when encountered in the header string.
Default: BLANK=YES

Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: YES specifies that the EBCDIC blank character (X'40') is to be ignored by the macro whenever characters are being skipped or whenever it is encountered in the header character string being checked against the skip character string specified by the *skipchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when characters are being skipped or whenever it is encountered in the header string being compared to the string specified by *skipchars.*

char specifies that the single character replacing *char* is to be ignored by this macro whenever characters are being skipped or whenever it is encountered in the header string being compared to the string specified by the *skipchars* operand. That is, the macro automatically skips over the character without counting or performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header, but is counted or compared to the character in the corresponding space in the *skipchars* string, like any other character.

POINT={BACK / FORWARD}

Function: Specifies whether the scan pointer is to be moved forward or backward.
Default: POINT=FORWARD
Format: BACK or FORWARD.
Notes: FORWARD specifies that the scan pointer is to be moved forward. BACK specifies that the scan pointer is to be moved backward. If POINT=BACK is specified, neither *skipchars* nor MOVE=RETURN may be specified. If POINT=BACK is specified, the scan pointer may not be moved out of the buffer in which it is located; if *integer* is greater than the number of characters preceding the scan pointer in the buffer, the macro does not execute and a return code of X'04' is placed in the rightmost byte of register 15.

MOVE={RETURN / KEEP}

Function: Specifies whether the scan pointer is to be moved to the designated position and not returned to its original position before the next macro is issued, or is to remain stationary, with the specified character string being located and the main-storage address of the last character in the string being returned in a designated register.
Default: MOVE=KEEP
Format: RETURN or KEEP.
Notes: If *integer* is replaced by zero, this operand is ignored. If *integer* is specified, MOVE=KEEP is assumed by TCAM even if the macro is coded MOVE=RETURN.

MOVE=RETURN may not be specified if POINT=BACK is specified.

KEEP specifies that the scan pointer is to be moved to the designated character (if *skipchars* is coded) or moved the designated number of characters (if *integer* is coded); the pointer remains in its new location until the next macro affecting the scan pointer is issued.

RETURN specifies that the scan pointer is not to be moved. Instead, the specified character string is located and the main-storage address of the last character in the string is returned in the register designated by the RESULT= operand.

RESULT={(register) / (15)}

Function: Specifies the general register into which the main-storage address of the last character of the designated character string is to be placed once the string is located.
Default: RESULT=(15)
Format: A general register 2 through 11, or 15, enclosed in parentheses.

Notes: If the desired character string is found, the address of its last character is placed in the register. If RESULT=(15) is coded and the character string is not found in this buffer (if *skipchars* is coded) or if the integer specified would take the scan pointer out of this buffer (if *integer* is specified), the macro does not execute and a return code of X'00' is placed in the rightmost byte of register 15.

If some register other than 15 is specified and the character string is not found in the buffer, or the integer specified would take the scan pointer out of the buffer, the macro does not execute and a return code of X'04' is placed in the low-order byte of register 15; in this case the contents of the specified register are unchanged.

If *integer* is replaced by zero, this operand is ignored.

Example 1: A SETSCAN macro that causes the scan pointer to skip forward over 5 characters and remain in its new position:

```
SETSCAN 5,POINT=FORWARD,MOVE=KEEP
```

Example 2: A SETSCAN macro that results in no movement by the scan pointer, but causes the address of the character '=' (located to the right of the pointer) to be placed in register 2:

```
SETSCAN C'=',POINT=FORWARD,MOVE=RETURN,RESULT=(2)
```

TERRSET Macro Instruction

The TERRSET macro:

- Enables the user to set a bit in the message error record.
- Is often issued prior to a related ERRORMSG macro,
- Is optional in inheader, inbuffer, outheader and outbuffer subgroups.

The TERRSET macro enables the user to set on bit 20 of the message error record for this message. This may be used to indicate any condition for which the user wishes to issue an error message. A subsequent ERRORMSG macro may be issued specifying a mask including the user error bit.

Name	*Operation*	*Operand*
[symbol]	TERRSET	(no operands)

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

There are no operands.

UNLOCK Macro Instruction

The UNLOCK macro:

- Removes a station from extended lock mode,
- Is optional in an inheader or outheader subgroup.

The UNLOCK macro is included in the inheader subgroup of an MH to remove a station locked to an application program by a LOCK macro from extended lock mode. The *conchars* operand may be used in conjunction with control characters in the message header to provide the capability of optional execution of UNLOCK.

The UNLOCK macro is meaningful only in the inheader subgroup of an MH handling inquiry or response messages being exchanged by an application program and a station locked to it by a LOCK macro having EXTEND coded as an operand. When the lock condition is not in effect, the macro is ignored.

When the UNLOCK macro is issued in an inheader subgroup handling inquiry messages being received from a station in extended lock mode, the message currently being

handled is routed to the destination specified in its header or by a FORWARD macro (and checked by a FORWARD macro) if UNLOCK is issued before the FORWARD macro is issued. If UNLOCK is issued after FORWARD, the message is routed to the application program to which the originating station was locked.

The UNLOCK macro may be issued immediately following an unconditional LOCK macro to remove a certain message type from lock mode before the message is queued. For example,

```
LOCK EXTEND
UNLOCK A
```

would place the station in extended lock mode for all except type A messages. Again,

```
LOCK MESSAGE
UNLOCK J
```

would terminate message lock mode only for type J messages.

For a discussion of the lock mode and its function in a TCAM system, see the description of the LOCK macro.

The UNLOCK macro has the following format:

Name	*Operation*	*Operand*
[symbol]	UNLOCK	[conchars[,BLANK={YES / NO / char}]]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

conchars

Function: Specifies the character or character string that, if found in the header as the next nonblank field, causes execution of the function.
Default: None. Specification optional.
Format: One to eight nonblank characters in character or hexadecimal format. If character format is used, the string may be unframed or framed with C' ' or CLn' ' characters. If hexadecimal format is used the string must be framed with X' ' or XLn' ' characters.
Notes: If this operand is omitted, the UNLOCK function is performed unconditionally. If the next field in the header does not match this operand, the function is not performed.

BLANK={YES / NO / char}

Function: Specifies whether EBCDIC blank characters are to be ignored when encountered in the character string in the message header being compared to the string specified by the *conchars* operand, or whether blanks are to be part of the header string when encountered in it. If EBCDIC blanks are to be counted as part of the header string, this operand also specifies whether some other hexadecimal character is to be ignored when encountered in the header string.
Default: BLANK=YES
Format: YES, NO or char. *char* is a single character that may be specified in either character or hexadecimal format. If character format is specified, it may be unframed or framed with C' ' or CL1' ' characters. If hexadecimal format is specified, it must be framed with X' ' or XL1' ' characters.
Notes: This operand is meaningless unless the *conchars* operand is also specified.

YES specifies that the EBCDIC blank character (X'40') is to be ignored by this macro whenever it is encountered in the header character string being checked against the control character string specified by the *conchars* operand. For example, if BLANK=YES is coded and an eight-byte field in the header is being checked by this

macro, a blank appearing in the fifth byte of the field will be ignored and the sixth through ninth bytes will be considered to be the last four bytes of the field (assuming that no blanks are coded in the sixth through ninth bytes).

NO specifies that the EBCDIC blank character is to be treated like any other character when it is encountered by this macro in the header string being compared to the string specified by *conchars.*

char specifies that the single character replacing *char* is to be ignored by this macro whenever it is encountered in the header string being compared to the string specified by the *conchars* operand. That is, the macro automatically skips over the character without performing a comparison and goes on to check the next character in the header. If BLANK=char is coded and *char* is not the EBCDIC blank character, the EBCDIC blank is not ignored by this macro when it is encountered in the header string, but is compared to the character in the corresponding space in the *conchars* string, like any other character.

NOTE: UNLOCK specified in an inheader subgroup removes the lock condition for the source terminal. In an outheader subgroup the lock for the destination is removed.

This chapter describes the physical parts of the MCP and tells how to arrange these parts in relation to each other and how to assemble, linkage-edit, and execute a TCAM MCP. In addition, several sample MCPs are presented and explained.

Arranging the Sections of the MCP

An MCP has five sections. These are the activation and deactivation section (the INTRO, OPEN, READY, and CLOSE macros and some user code to determine whether INTRO executed satisfactorily), a data set definition section (the DCB macros defining the TCAM data sets and, if application programs are to be run in conjunction with the MCP, one PCB macro for each application program), a terminal and line control area section (those macros associated with the invitation lists and the terminal table), a Message Handler section (one or more Message Handlers), and an optional user routine section (closed, user subroutines called by an MH, as well as exit routines referred to by the INTRO macro, by DCB macros, and by the STARTMH macro). These sections may be coded in the order just given, or in any other order, except that the activation and deactivation Section must come first. Each MCP section and locations for coding directions are given below. The sample program in this chapter will conform to this organization.

NOTE 1: The PCB macro defining an application program must be included in the MCP. This macro is logically similar to a DCB macro, and may be included in the Data Set Definition Section, or it may be included in any other section, but should not be coded between OPTION macros, among macros defining the terminal table, or within an MH.

NOTE 2: The INVLIST macros defining invitation lists must follow the macros defining the terminal table.

Activation and Deactivation

- Initializes, activates and deactivates the MCP;
- Opens and closes the MCP data sets;
- Described in the chapter *Activating and Deactivating the MCP.*

Data Set Definition

- Defines the data sets for the TCAM line groups, message queues, and checkpoint and logging facilities;
- Described in the chapter *Defining the MCP Data Sets.*

Terminal and Line Control Area

- Defines the terminal table for the MCP;
- Defines the option fields associated with the terminal-table entries;
- Defines the invitation list for each line;
- Described in the chapter *Defining Terminal and Line Control Areas.*

Message Handler

- Determines the way in which each incoming and outgoing message is to be processed;
- Routes each message to its proper destination, if possible;
- Informs the user of errors occurring during message transmission, routing, and handling;
- Described in the chapter *Designing the Message Handler.*

User Routine

- Consists of closed user subroutines that may be entered from a MH, or by an exit specified by an INTRO, STARTMH, FORWARD, ERRORMSG, READY, or DCB macro.
- Described in the section *Including a Closed Subroutine* of the chapter *Designing the Message Handler;* the description of each macro operand specifying an exit tells what TCAM passes to the routine associated with that exit, and what TCAM expects the exit routine to return.

Assembling, Linkage-editing, and Executing the Message Control Program

The assembly, linkage-editing, and execution of a TCAM MCP is similar to the assembly linkage-editing, and execution of any other problem program running under OS. Sample job control statements are given in this section for these three steps; these statements are guidelines only.

Assembling an MCP

A typical control-card sequence for assembling a TCAM MCP is as follows:

```
//ASSEMBLY   JOB      MSGLEVEL=1
//STEP1      EXEC     ASMFC
//ASM.SYSIN  DD       *

MCP Source Deck
```

Linkage-editing an MCP

A typical control-card sequence for linkage-editing an MCP is as follows (in this case, the MCP load module is to be placed in a private library called SYS1.TCAMLIB that has previously been created by the user):

```
//LINKEDIT   JOB      MSGLEVEL=1
//STEP1      EXEC     PGM=IEWL,PARM='XREF,LIST,LET',
//                    REGION=96K
//SYSPRINT   DD       SYSOUT=A
//SYSUT1     DD       UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSLMOD    DD       DSNAME=SYS1.TCAMLIB,DISP=OLD
//SYSLIB     DD       DSNAME=SYS1.TELCMLIB,DISP=OLD
//SYSLIN     DD       *

MCP Object Module

                    NAME     TCAMPROG(R)
```

The SYSLIB DD statement names SYS1.TELCMLIB, a special library area defined at system generation time. TCAM's resident modules are located in SYS1.TELCMLIB.

The MCP load module resulting from the link-edit may be stored in SYS1.LINKLIB, or in a private library such as SYS1.TCAMLIB in the example.

Executing an MCP

The TCAM MCP is normally executed as the highest-priority task in the highest-priority partition or region in the system. The TCAM MCP is executed either by placing the appropriate job control statements in the card reader and using an OS Reader/Interpreter to place the job in the system, or by issuing at the console a START command referring to a cataloged procedure that contains the necessary job control statements. (Starting by a START command is discussed below.) The job control statements needed for the execute step are similar for both cases.

A typical control card sequence for executing an MCP is as follows (in this case, the MCP has two line group data sets containing three lines each, and a message queues data set residing on disk; no checkpoint or logging facility is included). The load-module form of the MCP is placed on SYS1.TCAMLIB by the linkage editor.

```
//EXECMCP    JOB      'EXECUTE MCP',MSGLEVEL=1
//GOSTEP     EXEC     PGM=TCAMPROG,REGION=100K
//STEPLIB    DD       DSNAME=SYS1.TCAMLIB,DISP=SHR
//DD1050     DD       UNIT=025
//          DD       UNIT=026
//          DD       UNIT=027
//DD2740     DD       UNIT=015
//          DD       UNIT=016
//          DD       UNIT=017
//DISKDD     DD       DSNAME=DISKDS,DISP=OLD
//SYSABEND   DD       SYSOUT=A
```

The DISKDD DD statement is for a message queues data set residing on disk; DISKDD is the name specified in the DDNAME= operand of the DCB macro for this data set, while DISKDS is the name of the data set as specified by the DSNAME operand of the IEDQDATA DD statement for the IEDQXA utility used to preformat disk message-queues data sets residing on disk.

Information on the DD statments for line group data sets and message queues data sets is found in the chapter *Defining the MCP Data Sets.*

The STEPLIB DD statement defines SYS1.TCAMLIB, the private library on which the MCP was placed by the linkage editor. If the linkage editor had placed the MCP in SYS1.LINKLIB, no such DD statement would be needed. As an alternate to the STEPLIB DD statement, a JOBLIB DD statement could define the private library. The JOBLIB statement would immediately follow the JOB statement, and would be coded as follows:

```
//JOBLIB        DD          DSNAME=SYS1.TCAMLIB,DISP=SHR
```

Defining the private library by a STEPLIB DD statement is necessary if the MCP is running under MVT and is to be started by a START command; a JOBLIB DD statement may not be included in a cataloged procedure in SYS1.PROCLIB, while a STEPLIB DD statement may be so included in an MVT system (see the next section).

Starting the MCP by a Cataloged Procedure The user may catalog his job control statements for the execute step by using the IEBUPDTE utility program to place the statements in SYS1.PROCLIB. (IEBUPDTE is described in the OS publication *Utilities*). In order to start or restart his MCP, the user need only issue a START command from the system console naming his cataloged procedure. (Use of the START command for this purpose is described in the OS publication *Operator's Guide.*)

In the following example, three procedures are cataloged. The first of these, named PROC1, causes an MCP, named MCP1 located in SYS1.LINKLIB, to be started. The second and third procedures, named PROC2 and PROC3, both cause another MCP, named MCP2 and located in a private library named SYS1.TCAMLIB, to be started. The difference between PROC2 and PROC3 is in the line configurations specified; PROC2 contains DD statements for one line group data set, while PROC3 contains DD statements for two line group data sets. In the initialization section of MCP2 the user has coded OPEN macros for both data sets, and has also coded DCB macros for both data sets. When PROC2 is used to start MCP2, the OPEN macros for the extra line group data set and the DCB macro for this data set do not execute since no DD statement is present for this data set—but control passes to the next instruction in each case, and the MCP will be started. Thus, by specifying his cataloged procedures, the user can choose between MCPs, or between different line configurations for the same MCP, at start-up time.

To get MCP2 with 2 line groups, at start-up time the user would enter

```
START PROC3.ID
```

at the system console (ID is the identification sequence that must be specified by TCAM MODIFY commands entered at the system console; see the section *Specifying Operator Control Messages* in *Using TCAM Service Facilities*).

```
//STARTPGM    JOB         MSGLEVEL=1
//           EXEC         PGM=IEBUPDTE
//SYSPRINT    DD          SYSOUT=A
//SYSUT1      DD          DSNAME=SYS1.PROCLIB,DISP=OLD
//SYSUT2      DD          DSNAME=SYS1.PROCLIB,DISP=OLD
//SYSIN       DD          DATA
./             ADD       NAME=PROC1,LIST=ALL
./             NUMBER    NEW1=10,INCR=20
//           EXEC         PGM=MCP1
//DD1050      DD          UNIT=015
//            DD          UNIT=016
```

```
//              DD      UNIT=017
//SYSABEND      DD      SYSOUT=A
./              ADD     NAME=PROC2,LIST=ALL
./              NUMBER  NEW1=10,INCR20
//              EXEC    PGM=MCP2
//STEPLIB       DD      DSNAME=SYS1.TCAMLIB,DISP=SHR
//DD2770        DD      UNIT=021
//              DD      UNIT=022
//              DD      UNIT=023
//SYSABEND      DD      SYSOUT=A
./              ADD     NAME=PROC3,LIST=ALL
./              NUMBER  NEW1=10,INCR=20
//              EXEC    PGM=MCP2
//STEPLIB       DD      DSNAME=SYS1.TCAMLIB,DISP=SHR
//DD1050        DD      UNIT=015
//              DD      UNIT=016
//              DD      UNIT=017
//DD2770        DD      UNIT=021
//              DD      UNIT=022
//              DD      UNIT=023
//SYSABEND      DD      SYSOUT=A
/               ENDUP
/*
```

SAMPLE MCPS

This section presents three sample message control programs (MCPs) together with associated application programs and JCL statements. The examples consist of an MCP to message switch between terminal types, an MCP with an application program to demonstrate inquiry and rapid response, and an MCP with two application programs showing both file updating with checkpoint coordination and message retrieval.

The programs are designed to run under MVT on a 512K IBM System/360 Model 50. The LKED procedure used in the programs for inquiry and rapid response and file updating has been modified to linkage edit modules to a private library named SYS1.TCAMLIB rather than to SYS1.LINKLIB, the standard linkage library.

The first two programs run in a single region; the third needs three regions. Terminal requirements are included in the explanation preceding each MCP. The application programs provided are guidelines only and therefore do not demonstrate real processing.

MESSAGE SWITCHING BETWEEN TERMINAL TYPES

This MCP is designed to switch messages between IBM 1050 Data Communications Systems. It assumes two nonswitched 1050s on a multipoint line and a switched 1050 on another line. The addressing and invitation characters used in the TERMINAL and INVLIST macros, and the unit addresses on the DD JCL statements are installation-dependent. The values specified in this sample program are to be used as guidelines only.

The MCP is written to run in two steps. The first step is an assembly creating an object deck. If the assembly is successful, the second step is a link-edit and go using the object deck obtained from the assembly.

In addition to message switching, this sample program permits use of the operator control facility. Operator commands may be entered either from the system console or from the 1050 terminal named NYC1.

The format of a message entered in the system depends on whether it is a message to be switched or an operator command. If it is an operator command it must begin with the four characters OPID. If it is a message to be switched, its format is:

leading data X *destination* = *data* EOT

Since the translation table is 1050, the destination name and the X must be entered in upper case. The message must end with an EOT character. Examples of messages entered and responses received are:

a. Entered at NYC2, a message to be switched

   X CHGO = *message data newline*EOT
   Received at CHGO, the response
   X CHGO = 0001*message data newline*EOT

b. Entered at NYC1, an operator command

   OPID D TP,PRITERM *newline*EOT
   Received at NYC1, the response
   IED041I PRIMARY=SYSCON

Since all of the required INTRO operands are not specified in the assembly, the WTOR message IED002A SPECIFY TCAM PARAMETERS will be received when the GO step is executed. A minimum response is S=C,U. Any other INTRO operands with short keyword equivalents may be altered at this time. Any operands not specified in the assembly but required for this execution (for instance, fewer cross-reference entries, a system interval, or removal of on-line test) may also be specified as part of the response to the WTOR.

Each section of the sample program is commented to provide an explanation of the macros used and the operands specified.

```
//ASMMSGSW JOB MSGLEVEL=1
// EXEC ASMFC,PARM.ASM='NOLOAD,DECK'
//ASM.SYSIN DD *
MSGSWTCH CSECT
         PRINT NOGEN
*****
*
** ACTIVATION AND DEACTIVATION SECTION
*
* THIS SECTION INITIALIZES THIS MESSAGE CONTROL PROGRAM (INTRO MACRO),
* OPENS THE MCP DATA SETS (OPEN MACROS), ACTIVATES THE MCP (READY
* MACRO), CLOSES THE MCP DATA SETS (CLOSE MACROS) AND DEACTIVATES THE
* PROGRAM (RETURN MACRO).  SIXTY BUFFER UNITS (LNUNITS + MSUNITS) ARE
* DEFINED, AND THE LENGTH OF EACH BUFFER UNIT IS SET AT 116 BYTES
* (KEYLEN). THE NUMBER OF UNITS PER BUFFER IS DEFINED IN THE DCB MACROS
* IN THE DATA SET DEFINITION SECTION. THE TYPE OF STARTUP ON INTRO HAS
* BEEN OMITTED FROM THE MACRO TO PERMIT ALTERNATE SPECIFICATION AND
* ADDITION OF OPERANDS AT EXECUTION.  TWO LINE GROUPS (CONSISTING
* OF ONE LINE EACH) AND A MESSAGE QUEUES DATA SET ON REUSABLE DISK
* ARE OPENED.
*
*****
TCAMINIT INTRO CPB=2,                      CHANNEL PROGRAM BLOCKS          *
               DISK=YES,                   DISK QUEUING UTILIZED           *
               PROGID=MESSAGE/SWITCHING,   PROGRAM IDENTIFICATION          *
               LNUNITS=40,                 UNITS ASSIGNED TO LINES         *
               MSUNITS=20,                 UNITS ASSIGNED TO MAIN STORAGE  *
               KEYLEN=116,                 SIZE OF BUFFER UNITS            *
               CROSSRF=2,                  CROSS REFERENCE--DEBUG AID      *
               TRACE=10,                   I/O TRACE--DEBUG AID            *
               DTRACE=100,                 SUBTASK TRACE--DEBUG AID        *
               CONTROL=OPID                ID SEQUENCE FOR OPERATOR
*                                          CONTROL MESSAGES
         LTR   15,15                       TEST IF INTRO EXECUTED
         BZ    OPENDISK                    PROPERLY
*
NOEXEC   ABEND 123,DUMP                    IF NOT, ABNORMAL EXIT
*
OPENDISK OPEN  (DISK,(INOUT))              OPEN DISK QUEUE BEFORE LINES
         TM    DISK+48,DCBOFLGS            OPEN SUCCESSFUL
         BNO   NOEXEC                      NO - ABEND
*
OPENLINE OPEN  (GROUPONE,(INOUT),GROUPTWO,(INOUT)) OPEN LINE GROUPS
         TM    GROUPONE+48,DCBOFLGS        OPEN SUCCESSFUL
         BNO   NOEXEC                      NO - ABEND
*
         TM    GROUPTWO+48,DCBOFLGS        OPEN SUCCESSFUL
         BNO   NOEXEC                      NO - ABEND
*
ALLSWELL READY                             BEGIN PROCESSING
FINISHUP CLOSE (GROUPONE,,GROUPTWO)        CLOSE LINE GROUPS BEFORE DISK
         CLOSE DISK                        CLOSE DISK DATA SET
         L     13,4(13)                    RETURN CONTROL TO OS
         RETURN (14,12)                    SUPERVISOR
*
*****
*
** DATA SET DEFINITION SECTION
*
* THIS SECTION DEFINES THE DATA SETS FOR THE TCAM LINE GROUPS AND THE
* MESSAGE QUEUES ON DISK.  THE DISK MESSAGE QUEUES DATA SET IS DEFINED
* TO BE REUSABLE.  BOTH LINE GROUPS ARE IBM 1050 DATA COMMUNICATION
* SYSTEM GROUPS.  DYNAMIC BUFFER ALLOCATION IS NOT SPECIFIED FOR
* EITHER GROUP.  THEY BOTH USE THE SAME MESSAGE HANDLER (MH).  BUFFERS
* ARE BUILT WITH SINGLE BUFFER UNITS (BUFSIZE=116, AS IS KEYLEN ON
* THE INTRO MACRO).
*
*****
DISK     DCB   DSORG=TQ,                   ORGANIZATION IS TCAM DISK       *
               MACRF=(G,P),                REQUIRED OPERAND                *
               OPTCD=R,                    DATA SET ON REUSABLE DISK       *
               DDNAME=DISKDD               NAME OF ASSOCIATED DD JCL
*                                          STATEMENT
GROUPONE DCB   DSORG=TX,                   ORGANIZATION IS TCAM LINE       *
```

Figure 23. Sample Message-switching Program (Part 1 of 3)

```
                MACRF=(G,P),            REQUIRED OPERAND                *
                CPRI=E,                 SEND/RECEIVE PRIORITY EQUAL     *
                DDNAME=DDONE,           NAME OF DD JCL STATEMENT        *
                TRANS=1050,             1050 TRANSLATION TABLE          *
                SCT=1050,               SPECIAL CHARACTER TABLE         *
                MH=SWITCHMH,            MESSAGE HANDLER FOR LINE        *
                INVLIST=(LISTONE),      INVITATION LIST FOR LINE        *
                PCI=(N,N),              NO DYNAMIC ALLOCATION           *
                BUFSIZE=116,            SIZE OF A BUFFER                *
                BUFIN=2,                INITIAL ASSIGNMENT FOR INPUT    *
                BUFOUT=4,               INITIAL ASSIGNMENT FOR OUTPUT   *
                BUFMAX=4,               MAXIMUM BUFFERS PER LINE        *
                RESERVE=(21,0,0,0)      RESERVED FOR INSERTION OF
*                                       DATA IN MESSAGES
GROUPTWO DCB    DSORG=TX,               DCB FOR SECOND LINE GROUP       *
                MACRF=(G,P),                                            *
                CPRI=S,                 SEND HAS PRIORITY OVER RECEIVE  *
                TRANS=1050,                                             *
                SCT=1050,                                               *
                DDNAME=DDGRPTWO,                                        *
                MH=SWITCHMH,                                            *
                INVLIST=(LISTTWO),                                      *
                PCI=(N,N),                                              *
                BUFSIZE=116,                                            *
                BUFIN=2,                                                *
                BUFOUT=4,                                               *
                BUFMAX=4,                                               *
                RESERVE=(21,0,0,0)
*****
*
** TERMINAL AND LINE CONTROL AREA
*
* THIS SECTION DEFINES THE TERMINAL TABLE FOR THE MCP, THE ENTRIES
* IN THE TERMINAL TABLE, AND THE INVITATION LISTS FOR EACH LINE. THE
* TERMINALS NYC1 AND NYC2 ARE ASSOCIATED WITH THE LINE GROUP DEFINED
* BY THE GROUPONE DCB, WHILE CHGO IS THE ONLY LINE IN THE GROUPTWO
* LINE GROUP.  QUEUING IS BY TERMINAL FOR EACH TERMINAL, AND USES
* MAIN-STORAGE QUEUING WITH REUSABLE DISK BACKUP.  NYC1 IS DEFINED
* AS A SECONDARY OPERATOR CONTROL TERMINAL:  THUS NYC1 AND THE SYSTEM
* CONSOLE ARE THE ONLY TERMINALS CAPABLE OF ENTERING OPERATOR
* COMMANDS.  SINCE PRIMARY= WAS NOT SPECIFIED ON THE INTRO MACRO,
* THE SYSTEM CONSOLE IS THE PRIMARY OPERATOR CONTROL TERMINAL
* FOR THIS MCP.
*
*****
        TTABLE LAST=CHGO                THE LAST ENTRY IN THE TABLE
NYC1    TERMINAL QBY=T,                 QUEUING BY TERMINAL             *
                DCB=GROUPONE,           ASSOCIATED DCB                  *
                RLN=1,                  RELATIVE LINE NUMBER            *
                TERM=1050,              TYPE OF TERMINAL                *
                QUEUES=MR,              MAIN STORAGE, REUSABLE DISK     *
                ADDR=6402,              ADDRESSING CHARACTERS           *
                ALTDEST=NYC2,           ALTERNATE DESTINATION           *
                SECTERM=YES             SECONDARY OPERATOR CONTROL
NYC2    TERMINAL QBY=T,                 SECOND TERMINAL IN GROUP        *
                DCB=GROUPONE,                                           *
                RLN=1,                                                  *
                TERM=1050,                                              *
                QUEUES=MR,                                              *
                ADDR=62C2,                                              *
                ALTDEST=NYC1,                                           *
                SECTERM=NO              NOT AN OPERATOR CONTROL
*                                       TERMINAL
CHGO    TERMINAL QBY=T,                 TERMINAL FOR GROUPTWO           *
                DCB=GROUPTWO,                                           *
                RLN=1,                                                  *
                TERM=1050,                                              *
                QUEUES=MR,                                              *
                ADDR=62C2,                                              *
                DIALNO=NONE             MAY NOT BE CALLED BY THE
*                                       CENTRAL PROCESSING UNIT
*                                       BOTH ENTRIES ARE ACTIVE
LISTONE  INVLIST ORDER=(NYC1+640B,NYC2+620B) GROUPONE INVITATION LIST
LISTTWO  INVLIST ORDER=(CHGO+620B)      GROUPTWO INVITATION LIST
```

Figure 23. Sample Message-switching Program (Part 2 of 3)

```
*****
*
** MESSAGE HANDLER SECTION
*
* THIS SECTION PROVIDES THE MESSAGE-SWITCHING CAPABILITY OF THE MCP.
* INCOMING MESSAGES ARE TRANSLATED TO EBCDIC AND CHECKED TO SEE IF
* THEY ARE OPERATOR COMMANDS.  IF SO THEY ARE PROCESSED BY THE OPERATOR
* CONTROL FUNCTION.  IF THEY ARE NOT, THE DATE AND TIME IS INSERTED IN
* THE MESSAGE (USING 16 OF THE 21 BYTES RESERVED BY THE "RESERVE="
* OPERAND IN THE DCB) AND THE MESSAGE IS FORWARDED TO THE DESTINATION
* SPECIFIED IN THE HEADER.  OUTGOING MESSAGES ARE SEQUENCED (USING THE
* REMAINING RESERVED CHARACTERS), TRANSLATED BACK TO 1050 LINE CODE
* AND SENT.
*
*****
SWITCHMH STARTMH LC=IN                    LEAVE LINE CONTROL IN
*
* INCOMING GROUP OF THE MH
*
INGRCUP  INHDR                            PROCESS HEADERS ONLY
         CODE  ,                          TRANSLATE TO EBCDIC
         SETSCAN X                        SET SCAN POINTER TO AN X
         FORWARD (4)                      FORWARD TO THE DESTINATION
*                                         NAMED IN THE NEXT FOUR BYTES
*                                         OF THE MESSAGE
         INEND                            END OF THE INCOMING GROUP
*
* OUTGOING GROUP OF THE MH
*
OUTGRCUP OUTHDR                           PROCESS HEADERS ONLY
         SETSCAN =                        SET SCAN POINTER TO AN =
         SEQUENCE                         INSERT SEQUENCE NUMBER
         CODE  ,                          TRANSLATE TO 1050 LINE CODE
         OUTEND                           END OF THE OUTGOING GROUP
*
DCBOFLGS EQU   X'10'
         END



//LKEMSGSW JOB MSGLEVEL=1,REGION=120K,TYPRUN=HOLD
//S2 EXEC LKEDG
//LKED.SYSLIB DD DSN=SYS1.TCAMLIB,DISP=SHR
//LKED.SYSIN DD *

                    OBJECT DECK HERE

//GO.STEPLIB DD DSN=SYS1.TCAMLIB,DISP=SHR
//GO.DISKDD DD DSNAME=SAMP1,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//GO.DDCNE DD UNIT=015
//GO.DDGRPTWO DD UNIT=011
```

Figure 23. Sample Message-switching Program (Part 3 of 3)

INQUIRY AND RAPID RESPONSE

This MCP is designed to utilize the conversational capabilities of TCAM. It locks a terminal to an application program from the time a message is entered until a response is provided. A sample application program is also provided, but its functions are limited to recognition of messages. It does not do any processing.

The MCP assumes two nonswitched IBM 1050 Data Communications Systems on a multipoint line, and a single switched 1050 on another line. The addressing and invitation characters used in the TERMINAL and INVLIST macros, and the unit addresses on the DD JCL statements are installation-dependent. The values specified in the sample programs are guidelines only.

The programs are written to run in three steps. The MCP and the application program are are first assembled and object decks are obtained. If there are no assembly errors, the object decks are link-edited. The final step is the execution of the MCP, which will attach the application program after the MCP data sets are open.

The inquiry and rapid response feature has a limiting effect on transmission in the multipoint line group. If one of the terminals on the line enters a message, the other is locked out (cannot enter data) until the response has been received by the originating terminal. The terminal on the other line may enter messages during a lock on the multipoint line. The message format for the example is:

*origin*b *sequence*b = b*data*b/(16b) EOT

If the origin and sequence fields are valid, the response is:

origin date timeb*sequence*b = out-sequence b*data*b/

MESSAGE RECEIVED EOT

If the origin field was incorrectly specified, the response is:

ORIGIN FIELD WRONG

If the sequence number was incorrectly specified, the response is:

origin date timeb *sequence* SEQUENCE NUMBER HIGH

or

origin date timeb *sequence* SEQUENCE NUMBER LOW

Since the translation table used is 1050, the origin field must be entered in uppercase characters.

The MCP includes the operator control facility. Operator commands may be entered from the system console or from the terminal named NYC1. If an operator command is entered at NYC1, it must begin with the four-character identifier OPID.

Since all of the required INTRO operands are not specified in the assembly of the MCP, a WTOR message IED002A SPECIFY TCAM PARAMETERS will be received at the system console when the GO step is executed. The minimum required response is 'S=C,U'. Any other INTRO operands with short keyword equivalents may be altered, and operands not specified in the assembly but required for this execution may also be specified as part of the response to the WTOR.

Each section of the sample MCP and the application program that follows it has been commented to provide an explanation of the macros used and the operands specified.

```
//ASMING JOB MSGLEVEL=1
// EXEC ASMFC,PARM.ASM='NOLOAD,DECK'
//ASM.SYSIN DD *
INQUIRY  CSECT
         PRINT NOGEN
*****
*
** ACTIVATION AND DEACTIVATION SECTION
*
* THIS SECTION INITIALIZES THE MESSAGE CONTROL PROGRAM (INTRO MACRO),
* OPENS THE MCP DATA SETS (OPEN MACRO), ATTACHES THE APPLICATION
* PROGRAM (ATTACH MACRO), ACTIVATES THE MCP (READY MACRO), CLOSES THE
* MCP DATA SETS (CLOSE MACRO) AND DEACTIVATES THE PROGRAM (RETURN
* MACRO).  SIXTY BUFFERS (LNUNITS + MSUNITS) ARE DEFINED, AND THE
* LENGTH OF EACH BUFFER UNIT SET AT 116 (KEYLEN).  THE NUMBER OF UNITS
* PER BUFFER IS DEFINED IN THE DCB MACROS IN THE DATA SET DEFINITION
* SECTION.  THE TYPE OF STARTUP ON INTRO HAS BEEN OMITTED FROM THE
* MACRO TO PERMIT ALTERNATE SPECIFICATION AND ADDITION OF OPERANDS AT
* EXECUTION.  TWO LINES ARE OPENED.
*
*****
TCAMINIT INTRO DISK=NO,                    NO DISK QUEUING              *
               PROGID=INQUIRY/RESPONSE,    PROGRAM IDENTIFICATION       *
               LNUNITS=40,                 BUFFERS ASSIGNED TO LINES    *
               MSUNITS=20,                 BUFFERS ASSIGNED TO MAIN STOR *
               KEYLEN=116,                 SIZE OF BUFFER UNITS         *
               CROSSRF=2,                  CROSS-REFERENCE - DEBUG AID  *
               TRACE=10,                   I/O TRACE - DEBUG AID        *
               DTRACE=100,                 SUBTASK TRACE - DEBUG AID    *
               CONTROL=OPID                ID SEQUENCE FOR OPERATOR
*                                          CONTROL MESSAGES
         LTR   15,15                       TEST IF INTRO EXECUTED
         BZ    OPENLINE                    PROPERLY
*
NOEXEC   ABEND 123,DUMP                    IF NOT, ABNORMAL EXIT
*
OPENLINE OPEN  (GROUPONE,(INOUT),GROUPTWO,(INOUT)) OPEN LINE GROUPS
         TM    GROUPONE+48,DCBOFLGS        OPEN SUCCESSFUL
         BNO   NOEXEC                      NO - ABEND
*
         TM    GROUPTWO+48,DCBOFLGS        OPEN SUCCESSFUL
         BNO   NOEXEC                      NO - ABEND
*
         ATTACH EP=INQAP                   ATTACH APPLICATION PROGRAM
         READY                             BEGIN PROCESSING
FINISHUP CLOSE (GROUPONE,,GROUPTWO)        CLOSE LINE GROUPS
         L     13,4(13)                    RETURN CONTROL TO OS
         RETURN (14,12)                    SUPERVISOR
*
*****
*
** DATA SET DEFINITION SECTION
*
* THIS SECTION DEFINES THE DATA SETS FOR THE TCAM LINE GROUPS AND THE
* PROCESS CONTROL INTERFACE.  BOTH LINE GROUPS ARE IBM 1050 DATA
* COMMUNICATION SYSTEM GROUPS.  DYNAMIC BUFFER ALLOCATION IS NOT
* SPECIFIED FOR EITHER GROUP.  THEY BOTH USE THE SAME MESSAGE HANDLER
* (MH).  BUFFERS ARE BUILT WITH SINGLE BUFFER UNITS.  THE PROCESS
* CONTROL BLOCK REFERS TO A DIFFERENT MH.  SINCE THE APPLICATION
* PROGRAM GETS AND PUTS MESSAGES, THE BUFFER SIZE FOR THE PROCESS
* CONTROL BLOCK IS THE SAME AS THAT FOR THE LINES.
*
*****
GROUPONE DCB   DSORG=TX,                   ORGANIZATION IS TCAM LINE    *
               MACRF=(G,P),                REQUIRED OPERAND             *
               CPRI=E,                     SEND/RECEIVE PRIORITY EQUAL  *
               DDNAME=DDGRPONE,            NAME OF DD JCL STATEMENT     *
               TRANS=1050,                 1050 TRANSLATION TABLE       *
               SCT=1050,                   SPECIAL CHARACTERS TABLE     *
               MH=LINEMH,                  MESSAGE HANDLER FOR LINE     *
               INVLIST=(LISTONE),          INVITATION LIST FOR LINE     *
               PCI=(N,N),                  NO DYNAMIC BUFFER ALLOCATION *
               BUFSIZE=116,                SIZE OF A BUFFER             *
               BUFIN=2,                    INITIAL ASSIGNMENT - INPUT   *
```

Figure 24. Sample Inquiry/Response Program (Part 1 of 6)

```
                BUFCUT=4,                 INITIAL ASSIGNMENT - OUTPUT     *
                BUFMAX=4,                 MAXIMUM BUFFERS PER LINE        *
                RESERVE=(21,C,0,0)        RESERVED FOR INSERTION OF
*                                         DATA IN MESSAGES
GROUPTWO DCB    DSORG=TX,                 DCB FOR SECOND LINE GROUP       *
                MACRF=(G,P),                                              *
                CPRI=S,                   SEND HAS PRIORITY OVER RECEIVE  *
                DDNAME=DDGRPTWO,                                          *
                TRANS=1050,                                               *
                SCT=1050,                                                 *
                MH=LINEMH,                                                *
                INVLIST=(LISTTWO),                                        *
                PCI=(N,N),                                                *
                BUFIN=2,                                                  *
                BUFCUT=4,                                                 *
                BUFMAX=4,                                                 *
                RESERVE=(21,0,0,0)
QPROC    PCB    MH=APPMH,                 PROCESS CONTROL BLOCK           *
                BUFSIZE=116,                                              *
                BUFIN=5,                                                  *
                BUFCUT=5,                                                 *
                RESERVE=(5,0)
*****
*
** TERMINAL AND LINE CONTROL SECTION
*
* THIS SECTION DEFINES THE TERMINAL TABLE FOR THE MCP, THE ENTRIES IN
* THE TERMINAL TABLE, AND THE INVITATION LISTS FOR EACH LINE.  THE
* TERMINALS NYC1 AND NYC2 ARE ASSOCIATED WITH THE LINE GROUP DEFINED
* BY THE GROUPONE DCB, WHILE CHGO IS THE ONLY LINE IN THE GROUPTWO
* LINE GROUP (THIS IS A SWITCHED LINE, GROUPONE IS NON-SWITCHED).
* QUEUING IS BY TERMINAL FOR EACH TERMINAL, AND USES MAIN-STORAGE ONLY
* QUEUING.  NYC1 IS DEFINED AS A SECONDARY OPERATOR CONTROL TERMINAL.
* TWO PROCESS ENTRIES ARE ALSO DEFINED, ONE FOR GET PROCESSING AND
* THE OTHER FOR PUT.  QUEUING FOR THE GET PROCESSOR IS MAIN-STORAGE
* ONLY AND IT IS DEFINED AS ITS OWN ALTERNATE DESTINATION.
* BOTH PROCESS ENTRIES REFER TO THE SAME PROCESS CONTROL BLOCK (PCB).
*
*****
         TTABLE LAST=CHGO                 LAST ENTRY IN THE TABLE
GPRC     TPROCESS PCB=QPROC,              PCB NAME                        *
               QUELES=MO,                 MAIN-STORAGE ONLY QUEUING       *
               ALTDEST=GPRC               ALTERNATE DESTINATION
PPRC     TPROCESS PCB=QPROC               PUT PROCESS ENTRY
NYC1     TERMINAL QBY=T,                  QUEUING BY TERMINAL             *
               DCB=GROUPONE,              ASSOCIATED DCB                  *
               RLN=1,                     RELATIVE LINE NUMBER            *
               TERM=1050,                 TYPE OF TERMINAL                *
               QUELES=MO,                 MAIN-STORAGE ONLY QUEUES        *
               ADDR=6402,                 ADDRESSING CHARACTERS           *
               NTBLKSZ=(116),             SIZE OF A BLOCK                 *
               SECTERM=YES                SECONDARY OPERATOR CONTROL
*                                         TERMINAL
NYC2     TERMINAL QBY=T,                 SECOND TERMINAL IN GROUPONE      *
               DCB=GROUPONE,                                              *
               RLN=1,                                                     *
               TERM=1050,                                                 *
               QUELES=MO,                                                 *
               ADDR=6202,                                                 *
               NTBLKSZ=(116),                                             *
               ALTDEST=NYC1,              ALTERNATE DESTINATION           *
               SECTERM=NO                 NOT A SECONDARY OPERATOR
*                                         CONTROL TERMINAL
CHGO     TERMINAL QBY=T,                 TERMINAL FOR GROUPTWO            *
               DCB=GROUPTWO,                                              *
               RLN=1,                                                     *
               TERM=1050,                                                 *
               QUELES=MO,                                                 *
               ADDR=62C2,                                                 *
               NTBLKSZ=(116),                                             *
               DIALNO=NONE                MAY NOT BE CALLED BY THE
*                                         CENTRAL PROCESSOR
LISTONE  INVLIST ORDER=(NYC1+6408,NYC2+6208) GROUPONE INVITATION LIST
LISTTWO  INVLIST ORDER=(CHGO+6208)        GROUPTWO INVITATION LIST
```

Figure 24. Sample Inquiry/Response Program (Part 2 of 6)

```
*****
*
** MESSAGE HANDLER SECTION
*
* THIS SECTION PROVIDES THE MESSAGE HANDLING FUNCTION OF THE MCP.
* IT CONTAINS TWO MHS.  THE FIRST RECEIVES INPUT FROM LINES AND
* FORWARDS TO THE GET APPLICATION PROGRAM AFTER ITS ORIGIN AND
* SEQUENCE NUMBER HAVE BEEN VERIFIED AND THE DATE AND TIME HAVE BEEN
* INSERTED.  THE TERMINAL THAT SENT THE MESSAGE IS LOCKED TO THE
* APPLICATION PROGRAM UNTIL A RESPONSE IS RECEIVED.  MESSAGES WITH
* INVALID ORIGIN OR SEQUENCE NUMBER ARE CANCELED, AND AN ERROR
* MESSAGE BUILT.  OUTGOING MESSAGES ARE SEQUENCED AND AN EOB/EOT IS
* INSERTED AT THE END OF THE MESSAGE.  THE SECOND MH RECEIVES INPUT
* FROM THE APPLICATION PROGRAM AND FORWARDS IT TO THE DESTINATION
* PROVIDED BY THE APPLICATION PROGRAM.
*
*****
*
* FIRST MESSAGE HANDLER - FOR LINES
*
LINEMH   STARTMH LC=OUT,                  REMOVE LINE CONTROL            *
               STOP=YES,                  STOP ON ERRORS                 *
               CONV=YES                   CONVERSE MODE REQUESTED
         INHDR                            TO PROCESS HEADERS
         CODE  1050                       CONVERT TO EBCDIC FROM 1050
*                                         LINE CODE
         ORIGIN 4                         GET FOUR-CHARACTER ORIGIN
*                                         FROM MESSAGE
         DATETIME                         INSERT DATE AND TIME
         SEQUENCE                         VERIFY SEQUENCE NUMBER
         FORWARD DEST=C'GPRC'             FORWARD TO GET PROCESSOR
         LOCK  MESSAGE                    LOCK TERMINAL TO APPLICATION
*                                         PROGRAM UNTIL RESPONSE
         INMSG                            FULL MESSAGE RECEIVED
         CANCELMG X'5800C00000'           CANCEL MESSAGES WITH INVALID
*                                         ORIGINS AND SEQUENCE NUMBERS
         MSGGEN X'4000C00000',            IF ORIGIN WRONG, SEND          *
               CL18'ORIGIN FIELD WRONG'   THIS MESSAGE
         ERRORMSG X'1000000000',          IF SEQUENCE HIGH, RETURN THIS  *
               DEST=ORIGIN,               MESSAGE WITH THE HEADER TO ITS *
               DATA=C'SEQUENCE NUMBER HIGH' ORIGIN
         ERRORMSG X'08C0000000',          IF SEQUENCE LOW, RETURN THIS   *
               DEST=ORIGIN,               MESSAGE WITH THE HEADER TO ITS *
               DATA=C'SEQUENCE NUMBER LOW' ORIGIN
         INEND                            END OF INCOMING GROUP
         OUTHDR                           TO PROCESS HEADERS
         SETSCAN C'='                     SET SCAN POINTER TO AN =
         SEQUENCE                         INSERT SEQUENCE NUMBER
         CODE  ,                          TRANSLATE BACK TO LINE CODE
         MSGFORM                          INSERT EOB/EOT IN MESSAGE
         OUTEND                           END OF OUTGOING GROUP
*                                         THIS MH
*
* SECOND MESSAGE HANDLER - FOR APPLICATION PROGRAM
*
APPMH    STARTMH LC=OUT                   REMOVE LINE CONTROL
         INHDR                            TO PROCESS HEADERS
         FORWARD DEST=PUT                 TO DESTINATION IN HEADER
*                                         PROVIDED BY PUT APPLICATION
*                                         PROGRAM
         INEND                            END OF INCOMING GROUP
         OUTHDR                           PROCESS OUTGOING HEADERS
         SETSCAN C'='                     RESET SCAN POINTER
         SETEOF C'CLOSEAP'                SET END OF FILE IF CLOSEDOWN
*                                         MESSAGE RECEIVED
         OUTEND                           END OF OUTGOING GROUP
DCBOFLGS EQU   X'1C'
         END
```

Figure 24. Sample Inquiry/Response Program (Part 3 of 6)

```
//ASMINQAP JOB MSGLEVEL=1
// EXEC ASMFC,PARM.ASM='NOLOAD,DECK'
//ASM.SYSIN DD *
INQAP    CSECT
         PRINT NOGEN
*****
*
** INITIALIZATION SECTION
*
* THIS SECTION DOES THE NECESSARY INITIALIZATION FOR THE PROGRAM.
* IT SAVES REGISTERS, ESTABLISHES ADDRESSABILITY AND SETS THE
* NEW SAVE AREA ADDRESS IN THE STANDARD SAVE AREA REGISTER.
*
*****
         SAVE  (14,12),,*              SAVE REGISTERS
         LR    12,15                   RESET BASE REGISTER
         USING INQAP,12                ESTABLISH ADDRESSABILITY
         ST    13,SAVE+4               SAVE ADDRESS OF SAVE AREA
         LA    13,SAVE                 NEW SAVE AREA ADDRESS
*****
*
** ACTIVATION SECTION
*
* THIS SECTION OPENS THE DATA SETS FOR THE PROGRAM.
*
*****
OPEN     EQU   *
         OPEN  DCBIN                   OPEN INPUT DCB
         OPEN  DCBOUT                  OPEN OUTPUT DCB
*****
*
** PROCESSING SECTION
*
* THIS SECTION DOES THE ACCESS, PROCESSING AND RETURN OF MESSAGES.
* IT ALSO TESTS FOR THE NEED TO CLOSE DOWN THE PROGRAM, AND CLEARS
* WORK AREAS IF NOT.
*
*****
LOOP     EQU   *
         LA    10,GOTMSG               GET END OF WORK AREA ADDRESS
         GET   DCBIN,WORK              GET A MESSAGE
         LA    9,1                     SET LENGTH COUNTER
         LA    2,WORK+8                GET START OF MSG DATA
LOOP2    EQU   *
         CLI   0(2),C'/'               SEARCH FOR END OF DATA
         BE    PUT                     IF FOUND, COMPLETE PROCESSING
*
         LA    9,1(9)                  INCREMENT LENGTH
         LA    2,1(2)                  BUMP TO NEXT CHARACTER
         CR    2,10                    END AND NO /
         BE    CLOSEM                  YES - CLOSE DOWN
*
         B     LOOP2                   CONTINUE SEARCH
*
PUT      EQU   *
         LA    2,1(2)                  GET PAST LAST CHARACTER
         MVC   0(16,2),GOTMSG          PUT 'MSG RECEIVED' IN MSG
         LA    9,27(9)                 ADD INSERT LENGTH
         STH   9,DCBOUT+82             PUT LENGTH IN LRECL FIELD.
         PUT   DCBOUT,WORK             PUT THE MESSAGE
         MVI   WORK,C' '
         MVC   WORK+1(149),WORK        CLEAR WORK AREA TO BLANKS
         B     LOOP                    GET ANOTHER MESSAGE
*
*****
*
** DEACTIVATION SECTION
*
* THIS SECTION CLOSES THE DATA SETS FOR THE PROGRAM AND RETURNS TO
* THE OS SUPERVISOR
*
*****
CLOSEM   EQU   *
         CLOSE DCBIN                   CLOSE INPUT DCB
```

Figure 24. Sample Inquiry/Response Program (Part 4 of 6)

```
         CLOSE DCBOUT                  CLOSE OUTPUT DCB
         L     13,SAVE+4               RESTORE ADDRESS OF SAVE AREA
         RETURN (14,12)                RETURN TO OS SUPERVISOR
*
*****
*
** ERROR HANDLING SECTION
*
* THIS SECTION PROVIDES THE ERROR HANDLING FOR UNCORRECTABLE I/O
* ERRORS AND END OF DATA.
*
*****
ERROR    EQU   *
         WTO   'SYNAD ENTERED'         UNCORRECTABLE ERROR
         B     CLOSEM                  CLOSE DOWN THE PROGRAM
*
END      EQU   *
         WTO   'EODAD ENTERED'         END OF DATA INDICATOR
         B     CLOSEM                  CLOSE DOWN THE PROGRAM
*
*****
*
** DATA SET DEFINITION SECTION
*
* THIS SECTION DEFINES THE DATA SETS USED BY THE PROGRAM.
*
*****
DCBIN    DCB   DSORG=PS,               PHYSICAL SEQUENTIAL           *
               BLKSIZE=124,            SIZE OF MESSAGE AND WORK      *
               DDNAME=APPIN,           NAME OF DD JCL STATEMENT      *
               SYNAD=ERROR,            UNCORRECTABLE ERROR HANDLER   *
               EODAD=END,              END OF DATA HANDLER           *
               LRECL=116,              SIZE OF LOGICAL RECORD        *
               OPTCD=W,                BUILD PREFIX FOR SOURCE       *
               MACRF=GM                DCB FOR GET
DCBOUT   DCB   DSORG=PS,                                             *
               BLKSIZE=124,                                          *
               DDNAME=APPOUT,                                        *
               SYNAD=ERROR,                                          *
               LRECL=116,                                            *
               OPTCD=WU,                                             *
               MACRF=PM                DCB FOR PUT
*****
*
** WORK AREA DEFINITION SECTION
*
* THIS SECTION DEFINES THE WORK AREAS USED BY THE PROGRAM.
*
*****
SAVE     DC    18F'0'                  PROGRAM SAVE AREA
WORK     DC    150C' '                 WORK AREA FOR MESSAGE
GOTMSG   DC    C'MESSAGE RECEIVED'     MESSAGE PROCESSED INDICATOR
*
         END
```

Figure 24. Sample Inquiry/Response Program (Part 5 of 6)

```
//LKEDING JOB MSGLEVEL=1
// EXEC LKED
//LKED.SYSLMOD DD DSN=SYS1.TCAMLIB,DISP=OLD
//LKED.SYSIN DD *

                    OBJECT DECK HERE

 NAME INQUIRY(R)
//LKEDINQAP JOB MSGLEVEL=1
// EXEC LKED
//LKED.SYSLMOD DD DSN=SYS1.TCAMLIB,DISP=OLD
//LKED.SYSIN DD *

                    OBJECT DECK HERE

          NAME  INQAP(R)



//GOING JOB MSGLEVEL=1,REGION=120K,TYPRUN=HOLD
//JOBLIB DD DSN=SYS1.TCAMLIB,DISP=SHR
// EXEC PGM=INQUIRY
//GO.SYSABEND DD SYSOUT=A
//APPIN DD QNAME=GPRC
//APPOUT DD QNAME=PPRC
//DDGRPONE DD UNIT=029
//DDGRPTWO DD UNIT=021
```

Figure 24. Sample Inquiry/Response Program (Part 6 of 6)

FILE UPDATING WITH CHECKPOINT COORDINATION

This MCP demonstrates coordination of checkpointing by the MCP and an application program. It also has the capability to switch messages and to use the operator control facility. Finally, it has a second application program to utilize the retrieve capabilities of the POINT macro. Two lines are assumed, one a point-to-point line with two IBM 1050s, and the other supporting a single nonswitched IBM 2740. The addressing and invitation characters used in the TERMINAL and INVLIST macros, and the unit addresses on the DD JCL statements are installation-dependent. The values specified in the sample programs are guidelines only.

The job is set up to run in three steps. The MCP and both application programs are first assembled. Then object decks from the first step are link-edited. As the final step, the MCP is executed. The MCP will prompt at the system console when it is time for the application programs to be started.

The format of the message depends upon the function desired. If it is a message to be switched, the format on input is:

*destination*b S b*origin* b*data*bEOT

If the origin is correct, the message received at the destination is:

*destination*b S b*origin* b*data*

If the origin is invalid, the message received at the source will be:

ORIGIN FIELD WRONG

If it is a message for the application program that does the file update, the input format is:

*destination*b A b*sequence*b data b/(16b)EOT

For valid sequence numbers, the message received at the destination is:

out-sequence *destination*b A b*sequence* date time b*data*

If the sequence number is invalid, the message received at the source is:

SEQUENCE NUMBER HIGH

or

SEQUENCE NUMBER LOW

Messages for the retrieve application program may be formatted either:

*destination*b/b*data*b EOT

or

*destination*b A b*sequence*b/b*data*b EOT

The data is assumed to be in the format of the name of the origin of input messages or of the destination of output messages to be retrieved, a character I for input or O for output retrieval, and the sequence number of the message to be retrieved. Operator commands entered from NYC1 or CHGO must begin with the identifier OPID. Because of the translation tables used, messages entered by NYC1 and NYC2 must specify destination and origin in uppercase, while the same fields when entered by CHGO may enter these fields in either upper or lowercase.

Since all of the required INTRO operands were not specified in the assembly, the WTOR message IED002A SPECIFY TCAM PARAMETERS will be received at the console when the GO step is executed. A minimum response must specify some sort of restart with the S= operand. Any other operands with a short keyword equivalent specified in the assembly may be altered, and any operands not specified but required for this execution may also be specified as part of the response to the WTOR.

The MCP and its associated application programs are commented to provide an explanation of the macros used and the operands specified.

```
//ASMLPDT JOB MSGLEVEL=1
// EXEC ASMFC,PARM.ASM='LOAD,DECK'
//ASM.SYSIN DD *
LPDTCKPT CSECT
         PRINT NOGEN
*****
*
** ACTIVATION AND DEACTIVATION SECTION
*
* THIS SECTION INITIALIZES THE MESSAGE CONTROL PROGRAM (INTRO MACRO),
* OPENS THE MCP DATA SETS (OPEN MACROS), INDICATES THAT THE
* APPLICATION PROGRAMS MAY BE STARTED (WTO MACRO), ACTIVATES THE
* MCP (READY MACRO), CLOSES THE DATA SETS (CLOSE MACROS) AND
* DEACTIVATES THE PROGRAM (RETURN MACRO).  SIXTY BUFFERS (LNUNITS +
* MSUNITS) ARE DEFINED, AND THE LENGTH OF EACH BUFFER UNIT SET AT
* 116 (KEYLEN).  THE NUMBER OF UNITS PER BUFFER IS DEFINED IN THE
* DCE MACROS IN THE DATA SET DEFINITION SECTION.  THE TYPE OF
* STARTUP ON INTRO HAS BEEN OMITTED FROM THE MACRO TO PERMIT
* ALTERNATE SPECIFICATION AND ADDITION OF OPERANDS AT EXECUTION.
* TWO LINE GROUPS ARE OPENED.
*
*****
TCAMINIT INTRO CPB=2,                     CHANNEL PROGRAM BLOCKS          *
               DISK=YES,                  DISK QUEUING USED               *
               PROGID=CHECKPOINT/COORDINATION, PROGRAM IDENTIFICATION     *
               LNUNITS=40,                BUFFERS ASSIGNED TO LINES       *
               MSUNITS=20,                BUFFERS ASSIGNED TO MAIN STOR   *
               KEYLEN=116,                SIZE OF BUFFER UNITS            *
               CPINTVL=1800,              CHECKPOINT EVERY 30 MINUTES     *
               CKREQS=2,                  CKREQ MACROS IN FILEAP          *
               CROSSRF=2,                 CROSS-REFERENCE - DEBUG AID     *
               TRACE=10,                  I/O TRACE - DEBUG AID           *
               DTRACE=100,                SUBTASK TRACE - DEBUG AID       *
               CONTROL=OPID               ID SEQUENCE FOR OPERATOR
*                                         CONTROL MESSAGES
         LTR   15,15                      TEST IF INTRO EXECUTED
         BZ    OPENDISK                   IMPROPERLY
*
NOEXEC   ABEND 123,DUMP                   ABNORMAL EXIT
*
OPENDISK EQU   *
         OPEN  (DISK,(INOUT))             OPEN DISK QUEUE FIRST
         TM    DISK+48,DCBOFLGS           OPEN SUCCESSFUL
         BNO   NOEXEC                     NO - ABEND
*
         OPEN  (CKPT,(INOUT))             OPEN CHECKPOINT QUEUE NEXT
         TM    CKPT+48,DCBOFLGS           OPEN SUCCESSFUL
         BNO   NOEXEC                     NO - ABEND
*
         OPEN  (GROUPONE,(INOUT),GROUPTWO,(INOUT)) OPEN LINE QUEUES
         TM    GROUPONE+48,DCBOFLGS       OPEN SUCCESSFUL
         BNO   NOEXEC                     NO - ABEND
*
         TM    GROUPTWO+48,DCBOFLGS       OPEN SUCCESSFUL
         BNO   NOEXEC                     NO - ABEND
*
         WTO   'TIME TO START APPLICATION PROGRAMS'
         READY                            BEGIN PROCESSING
*
         CLOSE (GROUPONE,,GROUPTWO)       CLOSE LINE QUEUES FIRST
         CLOSE CKPT                       CLOSE CHECKPOINT QUEUE NEXT
         CLOSE DISK                       CLOSE DISK QUEUE LAST
         L     13,4(13)                   RETURN CONTROL TO OS
         RETURN (14,12)                   SUPERVISOR
*
*****
*
** DATA SET DEFINITION SECTION
*
* THIS SECTION DEFINES THE DATA SETS FOR THE TCAM DISK QUEUE,
* THE CHECKPOINT QUEUE, AND THE LINE GROUPS AND APPLICATION PROGRAM
* PROCESS CONTROL INTERFACE.  ONE LINE GROUP USES TWO IBM 1050 DATA
* COMMUNICATION SYSTEMS AND THE OTHER USES THE IBM 2740
* SYSTEM.  DYNAMIC BUFFER ALLOCATION IS NOT SPECIFIED FOR EITHER
```

Figure 25. Sample Checkpoint Coordination Program (Part 1 of 10)

```
* GROUP.  BOTH USE THE SAME MESSAGE HANDLER (MH), AND BOTH USE
* BUFFERS BUILT OF SINGLE UNITS.  THE PROCESS CONTROL BLOCKS
* REFER TO A DIFFERENT MH.  BUFFER SIZE FOR BOTH APPLICATION
* PROGRAMS IS THE SAME AS THAT FOR THE LINE GROUPS.
*
*****
DISK     DCB   DSORG=TQ,                ORGANIZATION IS TCAM DISK      *
               MACRF=(G,P),             REQUIRED OPERAND               *
               OPTCD=R,                 DATA SET ON REUSABLE DISK      *
               DDNAME=DISKDD            NAME OF DD JCL STATEMENT
CKPT     DCB   DSORG=TQ,                CHECKPOINT DATA SET            *
               MACRF=(G,P),                                            *
               OPTCD=C,                 DATA SET IS CHECKPOINT         *
               DDNAME=CKPTDD
GROUPONE DCB   DSORG=TX,                ORGANIZATION IS TCAM LINE      *
               MACRF=(G,P),                                            *
               CPRI=E,                  SEND/RECEIVE PRIORITY EQUAL    *
               DDNAME=DDONE,                                           *
               TRANS=1050,              1050 TRANSLATION TABLE         *
               SCT=1050,                SPECIAL CHARACTERS TABLE       *
               MH=LINEMH,               MESSAGE HANDLER FOR LINE       *
               INVLIST=(INVONE),        INVITATION LIST FOR LINE       *
               PCI=(N,N),               NO DYNAMIC BUFFER ALLOCATION   *
               BUFSIZE=116,             SIZE OF A BUFFER               *
               BUFIN=2,                 INITIAL ASSIGNMENT - INPUT     *
               BUFOUT=4,                INITIAL ASSIGNMENT - OUTPUT    *
               BUFMAX=4,                MAXIMUM BUFFERS PER LINE       *
               RESERVE=(21,0,0,0)       RESERVED FOR INSERTION OF
*                                       DATA IN MESSAGES
GROUPTWO DCB   DSORG=TX,                DCB FOR SECOND LINE GROUP      *
               MACRF=(G,P),                                            *
               CPRI=E,                                                 *
               TRANS=274F,              FOLDED 2740 TRANSLATION TABLE  *
               SCT=274F,                                               *
               DDNAME=DDTWO,                                           *
               MH=LINEMH,                                              *
               INVLIST=(INVTWO),                                       *
               PCI=(N,N),                                              *
               BUFSIZE=116,                                            *
               BUFIN=2,                                                *
               BUFOUT=4,                                               *
               BUFMAX=4,                                               *
               RESERVE=(21,0,0,0)
GPROC    PCB   MH=APPMH,                FILE PROCESS CONTROL BLOCK     *
               BUFSIZE=116,                                            *
               BUFIN=5,                                                *
               BUFOUT=5,                                               *
               RESERVE=(5,0)
RETRV    PCB   MH=APPMH,                PCB FOR RETRIEVE               *
               BUFSIZE=116,                                            *
               BUFIN=5,                                                *
               BUFOUT=5,                                               *
               RESERVE=(5,0)
*****
*
** TERMINAL AND LINE CONTROL SECTION
*
* THIS SECTION DEFINES THE TERMINAL TABLE FOR THE MCP, THE ENTRIES
* IN THE TERMINAL TABLE, AND THE INVITATION LISTS FOR EACH LINE.
* IT ALSO DEFINES AN OPTION FIELD TO MAINTAIN A COUNTER OF MESSAGES
* RECEIVED.  THE TERMINALS NYC1 AND NYC2 ARE ASSOCIATED WITH THE
* LINE GROUP DEFINED BY THE GROUPONE DCB, WHILE CHGO IS THE ONLY LINE
* IN THE GROUPTWO LINE GROUP.  QUEUING IS BY TERMINAL FOR EACH
* TERMINAL, AND MAIN-STORAGE QUEUING WITH REUSABLE DISK BACKUP IS
* USED.  NYC1 AND CHGO ARE BOTH DEFINED AS SECONDARY OPERATOR CONTROL
* TERMINALS.  FOUR PROCESS ENTRIES ARE ALSO DEFINED, TWO FOR GET AND
* TWO FOR PUT PROCESSING.  QUEUING FOR THE GET ENTRIES IS MAIN-
* STORAGE WITH REUSABLE DISK BACKUP.
*
*****
         TTABLE LAST=CHGO               LAST ENTRY IN THE TABLE
COUNTIN  OPTION H                       OPTION FIELD FOR COUNTER
FILE     TPROCESS PCB=GPROC,            PCB NAME                       *
               CKPTSYN=YES,             FOR CHECKPOINTING              *
```

Figure 25. Sample Checkpoint Coordination Program (Part 2 of 10)

```
              QUELES=MR,                MAIN-STORAGE, REUSABLE BACKUP  *
              ALTDEST=FILE              ALTERNATE DESTINATION
PUTF      TPROCESS PCB=QPROC            SECOND PROCESS ENTRY FOR FILEAP
GETR      TPROCESS PCB=RETRV,           PCB NAME                       *
              QUELES=MR,                                               *
              ALTDEST=GETR
PUTR      TPROCESS PCB=RETRV            SECOND PROCESS ENTRY FOR APRET
NYC1      TERMINAL CBY=T,               QUEUING BY TERMINAL            *
              DCB=GROUPONE,             ASSOCIATED DCB                 *
              RLN=1,                    RELATIVE LINE NUMBER           *
              TERM=1050,                TYPE OF TERMINAL               *
              QUELES=MR,                QUEUING TYPE                   *
              ADDR=64C2,                ADDRESSING CHARACTERS          *
              NTBLKSZ=(116),            SIZE OF A BLOCK                *
              OPDATA=0,                 INITIAL VALUE OF OPTION        *
              SECTERM=YES               SECONDARY OPERATOR CONTROL
*                                       TERMINAL
NYC2      TERMINAL CBY=T,               SECOND TERMINAL IN GROUP       *
              DCB=GROUPONE,                                            *
              RLN=2,                                                   *
              TERM=1050,                                               *
              ADDR=6202,                                               *
              QUELES=MR,                                               *
              OPDATA=0,                                                *
              NTBLKSZ=(116)
CHGO      TERMINAL CBY=T,               TERMINAL ON OTHER LINE GROUP   *
              DCB=GROUPTWO,                                            *
              RLN=1,                                                   *
              TERM=274F,                NONSWITCHED WITH CHECKING      *
              QUELES=MR,                                               *
              ADDR=E201,                                               *
              NTBLKSZ=(116),                                           *
              OPDATA=0,                                                *
              SECTERM=YES
INVONE    INVLIST ORDER=(NYC1+640B,NYC2+620B) GROUPONE INVITATION LIST
INVTWO    INVLIST ORDER=(CHGO+E201,CHGO+E201) GROUPTWO INVITATION LIST
*                                       POLL TWICE BEFORE DELAY
*****
*
** MESSAGE HANDLER SECTION
*
* THIS SECTION PROVIDES THE MESSAGE HANDLING FUNCTION OF THE MCP.
* IT CONTAINS TWO MHS.  THE FIRST RECEIVES INPUT FROM LINES AND
* FORWARDS TO THE DESTINATION SPECIFIED IN THE MESSAGE, WHICH MAY BE
* EITHER ANOTHER STATION OR AN APPLICATION PROGRAM.  MESSAGES ARE
* COUNTED, AND THE DATA INSERTED DEPENDS UPON A MESSAGE-TYPE
* INDICATOR SPECIFIED IN THE MESSAGE.  INVALID MESSAGES ARE
* CANCELED AND MESSAGES INDICATING THE ERROR ARE RETURNED TO THE
* ORIGINATING STATION.  THE SECOND MESSAGE HANDLER RECEIVES INPUT
* FROM EITHER OF THE APPLICATION PROGRAMS, SEQUENCES THEM AND
* RETURNS THEM TO THE DESTINATION SPECIFIED IN THE WORK AREA BUILT BY
* THE APPLICATION PROGRAM.
*
*****
LINEMH    STARTMH LC=OUT                TAKE OUT LINE CONTROL
          INHDR                         PROCESS INCOMING HEADERS
          CHECKPT                       CHECKPOINT OPTION FIELDS
          COUNTER COUNTIN               COUNT HEADERS RECEIVED
          CODE  ,                       TRANSLATE TO EBCDIC
          FORWARD DEST=**               FORWARD TO DESTINATION NAMED
*                                       IN NEXT FIELD OF MESSAGE
          MSGTYPE A                     TO AN APPLICATION PROGRAM
          SEQUENCE                      YES - SEQUENCE VERIFY IT
          DATETIME                      INSERT DATE AND TIME
          MSGTYPE S                     TO A SWITCHED TERMINAL
          ORIGIN ,                      YES - VERIFY ORIGIN
          INMSG                         TO PROCESS COMPLETE MESSAGE
          CANCELMG X'5800C00000'        CANCEL MESSAGES WITH INVALID
*                                       ORIGIN OR SEQUENCE NUMBER
          MSGGEN X'4000C0C000',         SEND INVALID ORIGIN MESSAGE    *
               CL18'ORIGIN FIELD WRONG' BACK TO WHOEVER SENT IT
          MSGGEN X'1000000000',         SEND SEQUENCE HIGH MESSAGE     *
               CL2C'SEQUENCE NUMBER HIGH' TO ITS SOURCE
          MSGGEN X'C800C00000',         SEND SEQUENCE LOW MESSAGE      *
```

Figure 25. Sample Checkpoint Coordination Program (Part 3 of 10)

```
                CL1S'SEQUENCE NUMBER LOW' TO ITS SOURCE
          INEND                          END OF INCOMING GROUP
          OUTHDR                         PROCESS OUTGOING HEADERS
          MSGFORM                        INSERT EOB/EOT AT END
          CODE  ,                        CONVERT BACK TO LINE CODE
          OUTEND                         END OF OUTGOING GROUP
*                                        THIS MH
APPMH     STARTMH LC=OUT                 REMOVE LINE CONTROL
          INHDR                          PROCESS INCOMING HEADERS
          FORWARD DEST=PUT               FORWARD TO DESTINATION PROVIDED
*                                        BY APPLICATION PROGRAM
          INEND                          END OF INCOMING GROUP
          OUTHDR                         PROCESS OUTGOING HEADERS
          SEQUENCE                       SEQUENCE OUTGOING MESSAGES
          OUTEND                         END OF OUTGOING GROUP
*
DCBOFLGS  EQU   X'1C'
          END
```

Figure 25. Sample Checkpoint Coordination Program (Part 4 of 10)

```
//ASMAPP1 JOB MSGLEVEL=1
// EXEC ASMFC,PARM.ASM='NOLOAD,DECK'
//ASM.SYSIN DD *
FILEAP   CSECT
         PRINT NOGEN
*****
*
** INITIALIZATION SECTION
*
* THIS SECTION ESTABLISHES ADDRESSABILITY AFTER SAVING THE CALLER
* REGISTERS.  A QSTART MACRO IS THE FIRST STATEMENT IN THE PROGRAM
* BECAUSE IT IS NEEDED IN ORDER TO USE THE CKREQ MACRO.
*
*****
         QSTART                          FOR CKREQ USAGE
         SAVE  (14,12),,*                SAVE REGISTERS
         LR    12,15                     SET BASE REGISTER
         USING FILEAP,12                 ESTABLISH ADDRESSABILITY
         ST    13,SAVE+4                 SAVE ADDRESS OF SAVE AREA
         LA    13,SAVE                   SET NEW SAVE AREA ADDRESS
*****
*
** ACTIVATION SECTION
*
* THIS SECTION OPENS ALL APPLICABLE DATA SETS.  IN THIS EXAMPLE, THE
* ONLY DATA SETS OPENED ARE THE TCAM DCBS.  IN A TRUE FILE UPDATING
* PROGRAM, THE DATA SETS FOR THE FILES WOULD ALSO BE OPENED IN THIS
* SECTION.
*
*****
         OPEN  DCBIN                     OPEN INPUT DCB
         OPEN  DCBOUT                    OPEN OUTPUT DCB
*****
*
** PROCESSING SECTION
*
* THIS SECTION DOES THE PROCESSING REQUIRED TO UPDATE FILES, AND TAKE
* THE COORDINATED CS AND TCAM CHECKPOINTS.  SINCE THIS IS ONLY AN
* EXAMPLE, NO FILES ARE UPDATED.  COMMENTS ARE PROVIDED TO EXPLAIN
* WHERE THE UPDATING AND CHECKPOINTING WOULD BE DONE IN A TRUE FILE
* UPDATING PROGRAM.  CHECKPOINTS ARE TAKEN AFTER EVERY 5 UPDATES.
*
*****
LOOP     EQU   *
         LA    2,5                       SET A LOOP CONTROL
         GET   DCBIN,WORK                GET A TCAM MESSAGE
         LA    5,GOTMSG                  GET END OF WORK AREA
         LA    4,1                       SET LRECL COUNT
         LA    3,WORK+8                  GET WORK AREA START
LOOP2    EQU   *
         CLI   0(3),C'/'                 SEARCH FOR END OF DATA
         BE    PUT                       FOUND - BUILD RESPONSE
*
         LA    3,1(3)                    BUMP TO NEXT BYTE
         LA    4,1(4)                    BUMP LRECL COUNTER
         CR    3,5                       END OF WORK
         BE    CLOSEM                    YES - ERROR
*
         B     LOOP2                     GO LOOK AT NEXT BYTE
PUT      EQU   *
         LA    3,1(3)                    GET PAST /
         MVC   0(16,3),GOTMSG            PUT RECEIVED IN MSG
         LA    4,27(4)                   INCREMENT LRECL COUNTER
         STH   4,DCBOUT+82               SET LRECL FIELD
         PUT   DCBOUT,WORK               PUT THE MESSAGE BACK TO THE
*                                         TCAM QUEUES
         BCT   2,LOOP                    DECREMENT AND TEST LOOP CONTROL
*
CSCKPT   BCR   0,0                       THE INSTRUCTIONS NEEDED TO CS
*                                         CHECKPOINT THE FILE JUST
*                                         UPDATED WOULD BE PLACED HERE
         CKREQ                           TCAM APPLICATION PROGRAM QUEUE
*                                         CHECKPOINT
CTEST    EQU   *
```

Figure 25. Sample Checkpoint Coordination Program (Part 5 of 10)

```
         WTOR  'TIME TO CLOSE  REPLY YES OR NO',REP,1,WECB
         XC    WECB(4),WECB              CLEAR ECB FOR A WAIT
         WAIT  ECB=WECB                  WAIT FOR RESPONSE
*
         CLI   REP,C'Y'                  REPLY YES
         BE    CLOSEM                    YES - CLOSE DOWN
*
         XC    REP(8),REP                CLEAR REPLY AREA
         MVI   WORK,X'40'
         MVC   WORK+1(149),WORK          CLEAR WORK AREA TO BLANKS
         B     LOOP                      GET ANOTHER MESSAGE
*
*****
*
** DEACTIVATION SECTION
*
* THIS SECTION DEACTIVATES THE DATA SETS USED BY THE PROGRAM.  ANY
* OTHER DATA SETS OPENED IN THE ACTIVATION SECTION WOULD BE CLOSED
* IN THIS SECTION.
*
*****
CLOSEM   EQU   *
         CLOSE DCBIN                     CLOSE INPUT DCB
         CLOSE DCBOUT                    CLOSE OUTPUT DCB
         L     13,SAVE+4                 RESTORE ADDRESS OF SAVE AREA
         RETURN (14,12)                  RETURN TO OS SUPERVISOR
*
*****
*
** ERROR HANDLING SECTION
*
* THIS SECTION PROVIDES THE ERROR HANDLING REQUIRED FOR
* UNCORRECTABLE ERRORS AND THE END-OF-DATA SITUATIONS.
*
*****
ERROR    EQU   *
         WTO   'SYNAD ENTERED'           UNCORRECTABLE ERROR
         B     CLOSEM                    CLOSE DOWN THE PROGRAM
*
END      EQU   *
         WTO   'EODAD ENTERED'           END OF DATA INDICATOR
         B     CTEST                     TEST IF CLOSEDOWN WANTED
*
*****
*
** CHECKPOINT SECTION
*
* THIS SECTION PROVIDES THE CHECKPOINTING AS SPECIFIED IN THE
* EXIT LIST OPERAND OF THE DCB MACROS.
*
*****
EXIT     EQU   *
         BCR   0,0                       IF OS CHECKPOINTING WERE
*                                        NEEDED (PER THE EXLST
*                                        OPERAND OF THE DCB MACROS)
*                                        THIS WOULD BE A ROUTINE TO DO
*                                        THE CHECKPOINTING
         CKREQ                           THIS COORDINATES WITH THE
*                                        TCAM CHECKPOINT
*****
*
** DATA SET DEFINITION SECTION
*
* THIS SECTION PROVIDES ONLY THE TWO TCAM DCBS.  ANY OTHER DCBS
* RELATIVE TO A FILE TO BE UPDATED WOULD BE DEFINED IN THIS SECTION.
*
*****
DCBIN    DCB   DSORG=PS,                 PHYSICAL SEQUENTIAL            *
               BLKSIZE=124,              SIZE OF MESSAGE AND WORK       *
               DDNAME=APPLIN,            NAME OF DD JCL STATEMENT       *
               SYNAD=ERROR,              UNCORRECTABLE ERROR HANDLER    *
               EODAD=END,                END OF DATA HANDLER            *
               EXLST=EXITLIST,           OS CHECKPOINT EXIT LIST        *
               LRECL=116,                SIZE OF LOGICAL RECORD         *
```

Figure 25. Sample Checkpoint Coordination Program (Part 6 of 10)

```
                OPTCD=W,                  BUILD PREFIX FOR SOURCE       *
                MACRF=GM                  DCB FOR GET
DCBOUT    DCB   DSORG=PS,                 OUTPUT DCB                    *
                BLKSIZE=124,                                          *
                DDNAME=APPLOUT,                                       *
                SYNAD=ERROR,                                          *
                EXLST=EXITLIST,                                       *
                LRECL=116,                                            *
                OPTCD=WU,                                             *
                MACRF=PM                  DCB FOR PUT
*****
*
*
** WORK AREA DEFINITION SECTION
* THIS SECTION DEFINES THE WORK AREAS USED BY THE PROGRAM.
*
*****
SAVE      DC    18F'0'                    SAVE AREA
REP       DC    2F'C'                     REPLY AREA FOR WTOR
WECB      DC    F'0'                      ECB FOR WTOR
WORK      DC    150C' '                   WORK AREA FOR MESSAGE
GOTMSG    DC    C'MESSAGE RECEIVED'       MESSAGE PROCESSED INDICATOR
EXITLIST  DC    X'8F'                     EXIT FOR CHECKPOINT
          DC    AL3(EXIT)                 ADDRESS OF CHECKPOINT ROUTINE
*
          END
```

Figure 25. Sample Checkpoint Coordination Program (Part 7 of 10)

```
//ASMAPP2 JOB MSGLEVEL=1
// EXEC ASMFC,PARM.ASM='NOLOAD,DECK'
//ASM.SYSIN DD *
RETRIEVE CSECT
         PRINT NOGEN
*****
*
** INITIALIZATION SECTION
*
* THIS SECTION PROVIDES THE NECESSARY INITIALIZATION FOR THE PROGRAM
* INCLUDING SAVING OF REGISTERS AND ESTABLISHING ADDRESSABILITY.
*
*****
         SAVE  (14,12),,*                SAVE REGISTERS
         LR    12,15                     RESET BASE REGISTER
         USING RETRIEVE,12               ESTABLISH ADDRESSABILITY
         ST    13,SAVE+4                 SAVE ADDRESS OF SAVE AREA
         LA    13,SAVE                   SET NEW SAVE AREA ADDRESS
*****
*
** ACTIVATION SECTION
*
* THIS SECTION OPENS THE DATA SETS USED IN THE PROGRAM.
*
*****
         OPEN  DCBIN                     OPEN DCB FOR INPUT
         OPEN  DCBOUT                    OPEN DCB FOR OUTPUT
*****
*
** PROCESSING SECTION
*
* THIS SECTION DOES THE PROCESSING NECESSARY TO DETERMINE FROM THE
* INPUT MESSAGE THE MESSAGE TO BE RETRIEVED, RETRIEVES IT AND SENDS
* IT BACK TO THE REQUESTER OF THE ORIGINAL MESSAGE.
*
*****
LOOP1    EQU   *
         LA    10,PTWORK                 GET END OF WORK AREA ADDRESS
         GET   DCBIN,WORK                GET REQUESTER MESSAGE
         LA    2,WORK+8                  GET START OF MESSAGE
LOOP2    EQU   *
         CLI   0(2),C'/'                 START OF DATA
         BE    PROCESS                   YES - PICK UP RETRIEVE DATA
*
         LA    2,1(2)                    BUMP TO NEXT CHARACTER
         CR    2,10                      END AND NO /
         BE    CLOSEM                    YES - CLOSE DOWN
*
         B     LOOP2                     CHECK FOR /
*
PROCESS  EQU   *
         MVC   TERMWORK(8),1(2)          PUT TERMNAME IN POINT WORK
         MVC   IOWORK(1),9(2)            PUT I OR O IN POINT WORK
         MVC   DOUBLE(5),10(2)           PUT SEQUENCE IN WORK AREA
         PACK  DOUBLE+6(2),DOUBLE(5)     CONVERT TO PACKED DECIMAL
         XC    DOUBLE(6),DOUBLE          CLEAR HIGH-ORDER BYTES
         CVB   3,DOUBLE                  CONVERT TO HEXADECIMAL
         STH   3,DOUBLE                  PUT IT BACK IN WORK AREA
         MVC   SEQWORK(2),DOUBLE         PUT INTO POINT WORK
         POINT DCBIN,PTWORK              POINT TO WORK AREA
         GET   DCBIN,WORK                GET RETRIEVAL RECORD
         MVC   WORK,TERMWORK             PUT NAME IN RETURN AREA
         LH    9,DCBIN+82                GET INPUT LRECL
         STH   9,DCBOUT+82               SET LRECL FOR PUT
         PUT   DCBOUT,WORK               RETURN RETRIEVED MESSAGE
CTEST    EQU   *
         WTOR  'TIME TO CLOSE  REPLY YES OR NO',REP,1,WECB
         XC    WECB(4),WECB              CLEAR ECB FOR WAIT
         WAIT  ECB=WECB                  WAIT FOR RESPONSE
*
         CLI   REP,C'Y'                  REPLY YES
         BE    CLOSEM                    YES - CLOSE DOWN THE PROGRAM
*
         XC    REP(8),REP                CLEAR RESPONSE AREA
```

Figure 25. Sample Checkpoint Coordination Program (Part 8 of 10)

```
         MVI   WORK,X'40'
         MVC   WORK+1(149),WORK        CLEAR WORK AREA TO BLANKS
         B     LOOP1                   GET ANOTHER RECORD
*
*****
*
** DEACTIVATION SECTION
*
* THIS SECTION DEACTIVATES THE DATA SETS USED IN THE PROGRAM AND
* RETURNS CONTROL TO THE OS SUPERVISOR.
*
*****
CLOSEM   EQU   *
         CLOSE DCBIN                   CLOSE INPUT DCB
         CLOSE DCBOUT                  CLOSE OUTPUT DCB
         L     13,SAVE+4               RESTORE ADDRESS OF SAVE AREA
         RETURN (14,12)                RETURN TO OS SUPERVISOR
*
*****
*
** ERROR HANDLING SECTION
*
* THIS SECTION PROVIDES THE ERROR HANDLING FOR UNCORRECTABLE ERRORS
* AND END OF DATA SITUATIONS.
*
*****
ERROR    EQU   *
         WTO   'SYNAD ENTERED'         UNCORRECTABLE ERROR
         B     CLOSEM                  CLOSE DOWN THE PROGRAM
*
END      EQU   *
         WTO   'EODAD ENTERED'         END OF DATA
         B     CTEST                   TEST IF CLOSEDOWN NEEDED
*
*****
*
** DATA SET DEFINITION SECTION
*
* THIS SECTION DEFINES THE DATA CONTROL BLOCKS FOR THE PROGRAM.
*
*****
DCBIN    DCB   DSORG=PS,               PHYSICAL SEQUENTIAL            *
               BLKSIZE=124,           SIZE OF MESSAGE AND WORK       *
               DDNAME=APP2IN,         NAME OF DD JCL STATEMENT       *
               SYNAD=ERROR,           UNCORRECTABLE ERROR HANDLER    *
               EODAD=END,             END OF DATA HANDLER            *
               LRECL=116,             SIZE OF LOGICAL RECORD         *
               OPTCD=W,               BUILD PREFIX FOR SOURCE        *
               MACRF=GMT              DCB FOR GET
DCBOUT   DCB   DSORG=PS,               OUTPUT DCB                     *
               BLKSIZE=124,                                          *
               DDNAME=APP2OUT,                                       *
               SYNAD=ERROR,                                          *
               LRECL=116,                                            *
               OPTCD=WU,                                             *
               MACRF=PM               DCB FOR PUT
*****
*
** WORK AREA DEFINITION SECTION
*
* THIS SECTION DEFINES THE WORK AREAS USED BY THE PROGRAM.
*
*****
SAVE     DC    18F'0'                  PROGRAM SAVE AREA
DOUBLE   DC    8C' '                   DOUBLEWORD WORK AREA
WECB     DC    F'0'                    ECB FOR WAIT
REP      DC    2F'C'                   AREA FOR WTOR REPLY
WORK     DC    150C' '                 AREA FOR GET AND PUT
PTWORK   EQU   *                       POINT WORK AREA
TERMWORK DC    8C' '                   FOR TERMINAL NAME
IOWORK   DC    C' '                    FOR I OR O
SEQWORK  DC    2X'C'                   FOR SEQUENCE NUMBER
*
         END
```

Figure 25. Sample Checkpoint Coordination Program (Part 9 of 10)

```
//LKEDPDT JOB MSGLEVEL=1
// EXEC LKED,PARM.LKED='LIST,LET,XREF'
//LKED.SYSLMOD DD DSN=SYS1.TCAMLIB,DISP=OLD
//LKED.SYSIN DD *

                    OBJECT DECK HERE

 NAME UPDTCKPT(R)
//LKEAPP1 JOB MSGLEVEL=1
// EXEC LKED
//LKED.SYSLMOD DD DSN=SYS1.TCAMLIB,DISP=OLD
//LKED.SYSIN DD *

                    OBJECT DECK HERE

 NAME FILEAP(R)
//LKEAPP2 JOB MSGLEVEL=1
// EXEC LKED
//LKED.SYSLMOD DD DSN=SYS1.TCAMLIB,DISP=OLD
//LKED.SYSIN DD *

                    OBJECT DECK HERE

 NAME RETRIEVE(R)



//GOLPDT JOB MSGLEVEL=1,TYPRUN=HOLD,REGION=120K
//JOBLIB DD DSN=SYS1.TCAMLIB,DISP=SHR
// EXEC PGM=UPDTCKPT
//SYSABEND DD SYSOUT=A
//DISKDD DD DSNAME=SAMP1,DISP=SHR
//CKPTDD DD DSNAME=SAMP2,UNIT=2311,VOLUME=SER=TSTAM1,SPACE=(TRK,(3)),
//             DISP=(NEW,CATLG)
//DDCRE DD UNIT=015
//DDTWC DD UNIT=017



//GOAPP1 JOB MSGLEVEL=1,TYPRUN=HOLD
//JOBLIB DD DSN=SYS1.TCAMLIB,DISP=SHR
// EXEC PGM=FILEAP
//APPLIN DD QNAME=FILE
//APPLOUT DD QNAME=PUTF



//GOAPP2 JOB MSGLEVEL=1,TYPRUN=HOLD
//JOBLIB DD DSN=SYS1.TCAMLIB,DISP=SHR
// EXEC PGM=RETRIEVE
//APP2IN DD QNAME=GETR
//APP2OUT DD QNAME=PUTR
```

Figure 25. Sample Checkpoint Coordination Program (Part 10 of 10)

WRITING TCAM-COMPATIBLE APPLICATION PROGRAMS

As described previously, a TCAM message may consist of header and text portions. The header portion is the primary concern of the Message Handler (MH) sections of the Message Control Program (MCP). If any processing of text portions of messages is required, this processing is performed by an *application program*, written by the user to suit the needs of his particular application. The main concern of TCAM with respect to an application program is to pass messages to the program for processing and later to return the messages to the appropriate station. (However, there may be no return message, as in the case of a file update application.) TCAM provides the means of transferring data between the patitions (GET, PUT, READ, WRITE, and CHECK macros), and provides a unique scheme for buffer usage for application programs. Application programs run asynchronously with the MCP, usually in another partition or region, but always as a separate system task or subtask. The MCP must have higher priority than any application program, since the MCP *must* have control after system interrupt (this becomes extremely important if the user's application program has a program loop that might cause continued contention with the MCP for control).

TCAM application programs need not be concerned with the station at which a message originated, or with the transmission code of the line, or with what the station line control had been. TCAM automatically handles line control in the Message Control Program. However, if a response message is generated, the application programmer must consider line control characters in the response, unless a MSGFORM macro is coded in the outheader subgroup handling messages for the destination station. The response message must be in line code unless the CODE macro is inserted in the outgoing group handling messages for the destination station.

Messages to be processed are placed in a destination queue by a Message Handler; a destination queue and its process entry in the terminal table are defined by a TPROCESS macro. A message from a station (or from an application program) can be routed to any predefined application program by a FORWARD macro.

The GET or READ macros that obtain messages from the destination queues transfer the data to a user-specified work area. The work area and the units of work placed in it are discussed below. Once in the work area, the data is analyzed and processed by the application program. Optionally, a PUT or a WRITE macro causes a response message to be returned to the Message Control Program for transmission either to a station (not necessarily the one that originated the message), to a list of destinations, or to another application program.

TCAM application programs allow the user to define at execution time, via the QNAME= parameter on the DD card, which of the destination queues specified in the terminal table is to be linked to the related data set.

TCAM allows the user to run his application programs in a non-teleprocessing environment for debugging, and then run them under TCAM without reassembling. The user may include such MCP-related, application-program TCAM macros as TCOPY, ICOPY, QCOPY, TCHNG, ICHNG, MRELEASE, and MCPCLOSE (all of which are discussed below) in an application program being debugged in a non-teleprocessing environment, provided that the macro definition library for the system under which the program is assembled includes the necessary macro definitions (as the result of a system generation procedure). When these macros are encountered at execution time in a system having no MCP, a return code is generated and control passes to the next instruction; otherwise, execution of the program is not hindered.

In some applications, the required processing may be such that one destination queue can handle all the messages, and a single application program having a single interface with the MCP can perform the processing. If various kinds of processing are required, there are two means of providing it:

- Each of several application programs may be provided with its own interface with the MCP, and the destination field in the message header is used to route the message to the appropriate destination queue for the desired program.

- Alternatively, all messages that require processing are routed to the same application program, where a user-written analysis routine determines the kind of message received, based upon a user-specified code in the message. The messages are transferred by this routine to the appropriate processing routines, or possibly to a processing program in another partition or region (via a PUT or WRITE back to the MCP).

When the destination field in the header is used to route messages to the appropriate processing program, the processing needed for the message must be determined with Message Handler facilities. Messages requiring different processing can be differentiated by MSGTYPE or PATH macros (see the descriptions of these macros).

Application programs transfer data to and from the MCP using GET/PUT (QSAM) or READ/WRITE/CHECK (BSAM) macro instructions. Support is provided for fixed-, variable-, and undefined-format work units. When using TCAM's GET/PUT support, the user may specify move or locate mode, but not substitute mode.

If the EODAD= operand is specified in the 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 after TCAM moves end-of-message into the user's work area. On succeeding GETs or READs, normal processing continues. If EODAD is not specified at end-of-data, the application program may stop issuing GETs or READs and issue a CLOSE macro to close the input DCB. If no SETEOF macro is issued, the GET or READ with CHECK is not finished until a message arrives on the queue. Time of entry to EODAD is controlled by the user because the real-time nature of the process queue for the application program. The SYNAD exit for logical errors is handled in the same manner as under BSAM and QSAM. The SYNADAF and SYNADRLS macros may be used.

Certain other features can also be incorporated into an application program:

1. A PUT or WRITE work area prefix can be used to specify the destination to which a message can be sent.
2. A GET or READ work area prefix can be used to receive the name of the message source.
3. The work area contents may be described to TCAM for PUT or WRITE operations and by TCAM for GET and READ operations as first segment, intermediate segment, last segment, or single-segment message.

These three options may be included at execution time by a DD card parameter (DCB=OPTCD=operand), or at assembly time by the appropriate DCB operands.

The POINT macro, used in conjunction with a GET or READ macro, provides the user with the capability to retrieve a message from a message-queues data set on disk, when this message has already been sent to its destination.

TCAM provides certain teleprocessing network control facilities from an application program (TCOPY, ICOPY, QCOPY, TCHNG, ICHNG, MRELEASE, and MCPCLOSE macros). All operator control functions are available from application programs; operator commands may be transferred by PUT or WRITE macros to the MCP. Responses to operator commands may be directed to any destination queue (except a PUT process entry) by the ALTDEST= operand of the PUT process entry.

Application programs written to run with a QTAM Message Control Program can be used when conversion is made from QTAM to TCAM. QTAM application programs being modified to run under TCAM need only be reassembled with a QSTART macro as the first instruction. During execution, the modified application program operates in most respects as it did under QTAM. *Appendix E* gives details on how to run QTAM application programs under TCAM.

Message Flow to an Application Program

This section describes the flow of a single-segment message between a remote station and an application program operating under TCAM with QSAM as the SAM interface. The steps described here are repeated for a multisegment message, except that the response message, if any, may be returned by the PUT macro any time after the first segment is received. This discussion summarizes the description in *Message Flow within the System* in the *TCAM Concepts and Facilities*, and adds a detail unique to application programs, the read-ahead queue.

A message segment enters the MCP and is placed in a buffer. The segment is handled by the incoming group of the MH for the originating station and is placed on the destination queue for the application program (called, hereafter, the *process queue*).

The segment is then removed from the destination queue and handled by the outgoing group of the MH for the application program. At this point, the message is queued on the read-ahead queue, an area in main storage related to the process queue. The read-ahead queue permits overlap of MCP and application-program processing of messages queued for a particular destination. This queue allows a message to be removed from a process queue to be processed by the outgoing group of the MH for the application program at the same time that a message that was previously on a process queue is being processed by the application program itself. The application program obtains the message from the read-ahead queue by GET or READ macro instructions. These macros obtain the messages in sections of data called *work units,* that will fit in an area of the application program called the *work area.* The message is placed in the work area for processing; the size of the work area bears no necessary relationship to the size of the MCP buffers.

After processing, and assuming there is a response message, the message is returned to the MCP, where it is placed in buffers. The buffers are handled by the incoming group of the MH for the application program and are placed on the appropriate destination queue (which may also be a process queue). After handling by the outgoing group of the MH for the destination, the response message is either sent on a line to a remote station or transferred to another application program.

Overview of the MCP/Application-Program Interface

The TCAM MCP routes messages between an application program and remote stations. Because an application program depends on the MCP to perform its input/output operations, an interface must be established between an application program and the MCP. TCAM provides the following services that establish interface from an application program:

- Definition of the interface (by the application program input and output DCB macros and DD statements, and by the PCB and TPROCESS macros in the MCP).
- Initialization and activation of the interface (by the OPEN macro).
- Transfer of messages between the application program and the MCP (by GET, PUT, READ, WRITE, CHECK, and POINT macro instructions).
- Deactivation of the interface (by the MCPCLOSE and CLOSE macros).

TCAM also provides buffer facilities specifically designed for the MCP interface.

Unlike the functions performed by the analysis and processing routines of an application program, these functions are partially or wholly peculiar to TCAM and the telecommunications environment. Therefore, TCAM provides routines to accomplish these functions. Linkage to these routines is established by TCAM and by standard data management macro instructions in an application program. These macros are discussed in this and succeeding sections.

Information necessary for communication between the MCP and an application program is provided by a control area defined by a PCB macro issued in the MCP (note also that the queues for an application program are defined by a TPROCESS macro in the MCP). No more than one application program can use a process control block, the control area defined by a PCB macro.

Message transfer from a destination queue to an application program is controlled by an input data control block (input DCB). An input DCB defines a logical data set called an input data set, which contains the messages being sent to the application program from a single destination queue created by a TPROCESS macro. If response messages are generated, message transfer from the application program to the MH queue is handled by another data control block, the output DCB. An output DCB defines a logical data set called an output data set, which contains messages being returned from the application program to the MCP by one process entry in the terminal table. (A PUT, GET, READ or WRITE macro names a DCB. The DCB macro specifies a DD statement. The QNAME parameter of the DD statement is coded with the name of a process entry. One data set must be defined for each process entry designed to receive messages from and send messages to an application program.) The user must define, open, and close the logical data sets represented by the DCBs.

A separate process entry must be specified for each input or output DCB in the application program. A DD statement must be provided for each such DCB. The format of the DD card is indicated later in this section.

Figure 26 shows how to set up the interface between the MCP and the application program by coding macro operands. Only those operands that help establish the interface are shown in the figure.

The GET, PUT, READ, WRITE, PCB, and input and output DCB macros, and the DD statements for the input and output DCB macros, are discussed in detail in this chapter. The TPROCESS macro is discussed in *Defining Terminal and Line Control Areas.*

The GET and PUT or READ and WRITE macros issued in an application program each specify the name of a data control block created by an input or output DCB macro. One input DCB macro must be coded in the application program for each terminal-table process entry named in a destination field in a message header or in an operand of the FORWARD macro to direct messages to the application program. A destination queue is created by TCAM for each such process entry. One output DCB macro must be coded in the application program for each process entry to be associated with response messages entered by the application program.

Each input or output DCB macro specifies (in its DDNAME= operand) a DD statement that must be included as part of the Job Control Language for execution of the application program. This DD statement has a QNAME= parameter that specifies the name of a process entry in the terminal table of the MCP. The TPROCESS macro that creates each process entry has a *pcbname* operand, which names a PCB macro. The PCB macro names an MH to handle messages being sent to or received from the application program by process entries whose TPROCESS macros name this PCB macro. The PCB macro is similar to the line group DCB macro in that both specify Message Handlers and other related values. The MH specified by the line group DCB macro handles messages transmitted between remote stations and the computer, while the MH specified by the PCB macro handles messages sent to and received from the application program by the MCP.

Defining the Components of the Interface

Among the components of the MCP/Application Program interface are the following:

- Process entries located in the terminal table and referred to by GET/READ and PUT/WRITE macros
- Data control blocks (and DD statements) for the application-program input and output data sets
- The process control block (this block specifies the MH for the application program)
- Buffers to transfer data between the MCP and the application-program work areas.

Process entries are created by TPROCESS macros (described in the chapter *Defining Terminal and Line Control Areas*). The other components of the interface are described in this section.

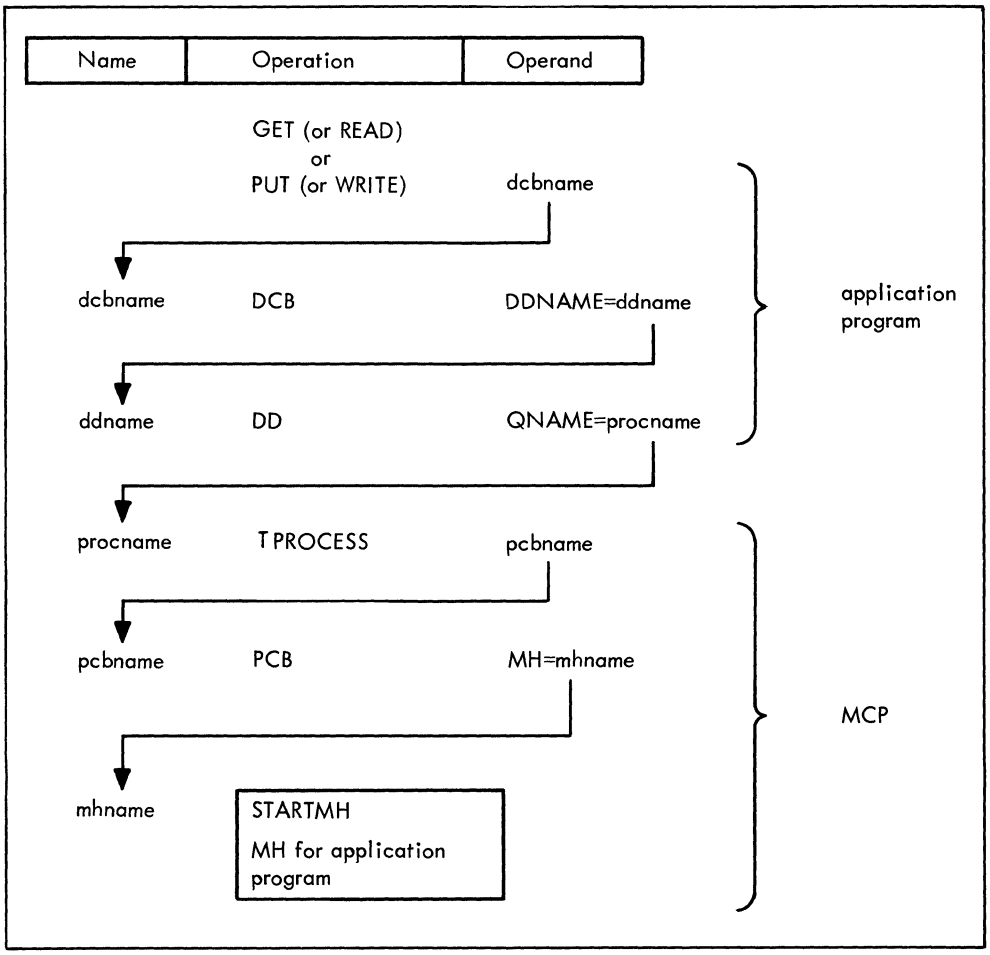


Figure 26. Interface between the Application Program and the MCP

Defining the Application Program Data Sets and the Process Control Block

Two types of logical data sets, called the input data set and the output data set, must be defined when writing a TCAM application program.

The input data set consists of the data (messages or records) sent to an application program from a single destination queue created by a TPROCESS macro (process queue). An input data set is defined by an input DCB macro. One input data set should be defined for each process queue.

The messages or records in an input data set are transferred from the process queue to the application program by a GET or READ macro that specifies the name of the input data set.

An output data set contains the messages or records returned from the application program to the MCP by a process entry in the terminal table. An output data set is defined by an output DCB macro. One output data set must be defined for each process entry designed to receive messages from an application program. Messages are transferred from the application program to the MCP by a PUT or WRITE macro specifying the name of the output data set.

The line group DCB macro for the MCP names the Message Handler that is to handle messages sent over any line in the line group for which it is issued. For the application program, this function is performed by the PCB macro rather than by the input or output DCB macro. One and only one PCB macro must be coded for each application

program that is to interface with the MCP. This macro is coded in the MCP rather than in the application program. In addition to assigning an MH to the application program, the PCB macro specifies the size of the buffers to be assigned by the MCP to handle messages being sent to and received from that application program.

The next sections describe the input and output DCB macros, the DD statements required for these macros, and the PCB macro. Many operands of the input and output DCB macros are concerned with aspects of data transfer and processing (type of record, type of work area, etc.); these operands should not be coded until *Transferring Data Between an MCP and an Application Program* in this chapter has been read.

Input DCB Macro Instruction

The input DCB macro:

- Defines an input data set for an application program;
- Must be issued for each process queue accessed by the application program with GET or READ macros;
- Specifies whether BSAM or QSAM is to be used 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.

The input DCB macro allocates main storage space for a data control block at assembly time. Parameters based on the operands specified in the macro are included in the data control block. The macro generates no executable code. One (and only one) input DCB macro is coded for each process queue to which the application program may direct a GET or a READ macro. (The GET or READ specifies the name of the input DCB macro; the DCB macro names a DD statement; the DD statement names a process entry in the terminal table.)

The input DCB macro has the following format:

Name	*Operation*	*Operands*
dcbname	DCB	keyword operands

dcbname

Function: Specifies the name of the macro instruction and also the name of the data control block generated by the expansion of the macro.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

keyword operands

Function: Specifies the operands that can be used.
Format: May be specified in any order, separated by commas with no intervening blanks.
Note: The operands are described below.

When a parameter can be provided by an alternate source, an appropriate symbol appears below the operand associated with that parameter. When there is no alternate source (i.e., the parameter must be specified by the operand), no symbol is shown. The symbols have the following meanings:

Symbol	*Explanation*
DD	The value of the operand can be omitted from the DCB macro and provided at execution time by the Data Definition (DD) card for the data set.
OE	The value of the operand can be provided by the problem program any time up to and including the data control block exit at open time.
PP	The value of the operand can be provided by the user's problem program any time before open time.

Note: If DD is specified, OE or PP may also be used. If OE is specified, PP may also be used.

For information on how to provide parameters by means of OE or PP, see *Data Management Services.* The same publication describes the data control block exit that can be taken when OE is specified. Information on providing parameters by DD is given in *DD Statements for the Input and Output Data Sets.*

DSORG=PS

Alternate Source: None.
Function: Specifies that the data control block governs message transfer to and from a destination queue, and identifies the data set organization as physical sequential.
Default: None. This operand is required.
Format: DSORG=PS
Notes: This operand achieves TCAM compatibility with QSAM or BSAM.

MACRF=({G{M / L} [T]} / {R [P]})

Alternate Source: None;
Function: Specifies the type of access to the destination queue.
Default: None. This operand is required.
Format: GM, GMT, GL, GLT, R, RP
Notes: G indicates GET (QSAM), R indicates READ in move mode (BSAM). GET is in move (M) or locate (L) mode.

The optional T indicates that the POINT macro may be used in conjunction with GET and may not be omitted if POINT is to be used with GET.

The optional P indicates that the POINT macro may be used in conjunction with READ and may not be omitted if POINT is to be used with READ.

If locate mode (L) is specified for a GET, TCAM obtains a work area by the GETMAIN macro instruction at OPEN time from the application program main storage. TCAM returns the address of the work area in register 1 following the first GET macro, and uses this work area for succeeding GETs (see *Dynamic Work-Area Definition* in this chapter). Locate mode is inconsistent with BSAM.

DDNAME=symbol

Alternate Source: PP.
Function: Specifies the name of the DD statement associated with the data control block.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If this operand is omitted, it must be provided by the alternate source.

BLKSIZE=integer

Alternate Source: DD.
Function: Specifies the length in bytes of the application program work area.
Default: None. This operand is required.
Format: Unframed decimal integer no smaller than the length of a record as specified by the LRECL= operand.
Maximum: 32760
Notes: If this operand is omitted, it must be provided by the alternate source.

The length of optional fields in the work area must be included in the value specified for this operand. TCAM uses this field to determine the length of the work area.

For undefined-format work units, the value specified for BLKSIZE= may be dynamically overridden on a work-unit-by-work-unit basis by the length operand of the READ macro.

BUFL=integer

Alternate Source: None.
Function: Specifies the size in bytes of buffers used in the MCP for messages coming to the application program associated with this DCB macro.
Default: None. Specification optional.
Format: Unframed decimal integer greater than 35.
Maximum: 65535
Notes: If this operand is omitted, the value specified in the BUFSIZE= operand of the PCB macro is used.

LRECL=integer

Alternate Source: DD.
Function: Specifies the number of bytes in the length of a record, plus the length of any optional fields in the work area.
Default: If RECFM=F, this operand is required. Otherwise, specification optional.
Format: Unframed decimal integer.
Maximum: 32760
Notes: If RECFM=U is specified, the LRECL= field in the input DCB is updated after each GET or READ macro with the sum of the number of bytes of data fetched by that GET or READ, plus the length of any optional fields in the work area.

RECFM={**F** / **V[B]** / **U**}

Alternate Source: DD.
Function: Specifies the format and characteristics of the work units in this input data set.
Default: RECFM=U
Format: F, V, VB, or U.
Notes: V specifies that the work units are variable in format. For BSAM and QSAM, each work unit is prefaced in the work area by a standard SAM four-byte prefix (all entries in the prefix are in hexadecimal format).

VB specifies that the work units are treated as blocked, although only one work unit is transferred per GET or READ. The variable-length work unit work area includes a blocked work area prefix of eight bytes if MACRF=R is specified, and of four bytes if not.

U specifies undefined-format work units. TCAM, like SAM, provides no prefix. The length of the work unit is stored by TCAM in the LRECL= field in the input DCB. TCAM updates the LRECL= field after each GET or READ with the length of the work unit.

F specifies fixed-length work units. The sum of the length of each work unit obtained plus the length of any optional fields in the work area is specified by the user in the LRECL= field of the input DCB and may be updated before each GET or READ. This option should be used only when the number of bytes of data in a message is an exact multiple of the number of bytes specified by the LRECL= operand. Otherwise, the last portion of the message contains fewer bytes than the number specified in the LRECL= operand, which the program must be capable of handling.

OPTCD=[W][U][C]

Alternate Source: DD.
Function: Specifies the optional fields for the work unit.
Default: None. Specification optional.
Format: W, WU, WC, WUC, U, UC, C.
Notes: W specifies that the name of the source of each message is to be placed in an eight-byte origin field in the work area. TCAM places the name of the source, in EBCDIC, in the field, left-adjusted and padded to the right with blanks. If W is coded but TCAM cannot determine the message source, the field is filled with eight character blanks.

U specifies that the work unit to be handled is either a message or a message segment that is not a record. If U is omitted, the work unit is assumed to be a record.

C specifies that a one-byte field in the work area, called the position field, is to indicate whether the work unit being handled is the first, an intermediate, or the last segment of the message. The contents of this field, filled in by TCAM, have the following meanings:

Position Field	*Work Area Contents*
X'F1' (1)	First portion of message
X'40' (blank)	Intermediate portion of message
X'F2' (2)	Last portion of message
X'F3' (3)	An entire message

EODAD=address

Alternate Source: PP.
Function: Specifies the address of an open or closed subroutine to be given control after the access method recognizes a user-generated end-of-file indication in the header of a message.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: TCAM takes this exit when the next GET or CHECK macro is issued following complete transfer of the end-of-file message into the work area.

SYNAD=address

Alternate Source: PP.
Function: Specifies the address of an open or closed subroutine to be given control if message processing is used, the work unit is larger than the work area, and OPTCD=C is not specified.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: For more information on the SYNAD exit, see *Application Program Error Exit* in this chapter.

EXLST=address

Alternate Source: PP.
Function: Specifies the address of the problem-program exit list.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: The list must start on a fullword boundary; its format and contents are more fully shown in *Data Management Services*. Each entry is a fullword made up of a control byte followed by the 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.

The entry having a control byte of X'05' is the DCB exit entry; it is explained in the *Data Management Services* publication.

If the control byte is X'0F', the user-written routine is given control to initiate an OS checkpoint of the application program (see the section on coordinating OS and TCAM checkpoints in this chapter).

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

Output DCB Macro Instruction

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 or BSAM is to be used 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.

The output DCB macro has the following format:

Name	*Operation*	*Operands*
dcbname	DCB	keyword operands

dcbname

Function: Specifies the name of the macro instruction and the name of the data control block operated by the expansion of the macro.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

keyword operands

Function: Specifies the operands that may be used.
Format: May be specified in any order, separated by blanks with no intervening commas.
Notes: The operands are described below.

When a parameter can be provided by an alternate source, an appropriate symbol appears below the operand associated with that parameter. When there is no alternate source (i.e., the parameter must be specified by the operand), no symbol is shown. The symbols have the following meanings:

Symbol	*Explanation*
DD	The value of the operand can be provided at execution time by the Data Definition (DD) card for the data set.
OE	The value of the operand can be provided by the problem program any time up to and including the data control block exit at open time.
PP	The value of the operand can be provided by the user's problem program any time before open time.

Note: If DD is specified, OE or PP may also be used. If OE is specified, PP may also be used.

For information on how to provide parameters by one of these alternate sources, see the note following the explanation of DD, OE, and PP in the discussion of the input DCB macro.

DSORG=PS

Alternate Source: None.
Function: Specifies that the data control block governs message transfer to or from an application program, and identifies the data set organization as physical sequential.
Default: None. This operand is required.
Format: DSORG=PS
Notes: This operand achieves TCAM compatibility with QSAM and BSAM.

DDNAME=symbol

Alternate Source: PP.
Function: Specifies the name that appears in the DD statement associated with the data control block.
Default: None: This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If this operand is omitted, it must be specified from the alternate source.

MACRF=$\left(\left\{\begin{matrix}P\left\{\begin{matrix}M\\L\end{matrix}\right\}\\W\end{matrix}\right\}\right)$

Alternate Source: None.
Function: Specifies the method by which messages are to be transferred to the destination queue.
Default: None. This operand is required.
Format: PM, PL or W.
Notes: P specifies that messages are to be transferred by PUT macros. W specifies that messages are to be transferred by WRITE macros.

PUT may be in move (M) or locate (L) mode. WRITE implies move mode.

If locate mode (L) is specified for PUT, TCAM obtains a work area by the GETMAIN macro instruction when the first PUT is executed. TCAM returns the address of the work area in register 1 following the first PUT (see *Dynamic Work-Area Definition* in this chapter).

BLKSIZE=integer

Alternate Source: DD.
Function: Specifies the length in bytes of the application program work area.
Default: None. If locate mode is not specified, specification optional. Otherwise, this operand is required.
Format: Unframed decimal integer no smaller than the length of a work unit.
Maximum: 32760
Notes: The length of any optional fields in the work area should be included in the value specified for this operand. If locate mode is specified by the MACRF= operand and this operand is omitted, it must be specified by an alternate source.

LRECL=integer

Alternate Source: None.
Function: Specifies the sum of the number of bytes in the length of a fixed- or undefined-length work unit, plus the length of any optional fields in the work area.
Default: If RECFM=F is specified, this operand is required. Otherwise, specification optional.
Format: Unframed decimal integer.
Maximum: 32760
Notes: If RECFM=U is specified and no work-unit length is specified by the length operand of the WRITE macro, the contents of the field must be updated dynamically by the program before a PUT or WRITE macro is issued; user code must place the number of bytes of data in the work area (including optional fields) in the LRECL= field of the DCB. This may be done with the aid of the DCBD macro, described in *Supervisor and Data Management Macro Instructions.*

If a value is specified by the length operand of the WRITE macro, this value overrides the value specified in the LRECL= field for undefined work units.

OPTCD=[W] [U] [C]

Alternate Source: DD.
Function: Specifies the type of optional field to be used.
Default: None. Specification optional.
Format: W, WU, WC, WUC, U, UC, C.
Notes: W specifies that the program must place the name of the destination of the message in an eight-byte destination field in the work area before a PUT or WRITE macro is executed. If a FORWARD macro with the operand DEST=PUT is coded in the incoming group of the application-program Message Handler, the message is routed to the destination specified in this field.

U specifies that the work unit is a message or a portion of a message that is not a record; if U is omitted, the work unit is assumed to be a record.

C specifies that a one-byte position field in the work area is used to describe the position of the work unit in the message of which it is a part. The control byte is defined as follows:

Position Field	*Work Area Contents*
X'F1' (1)	First portion of message
X'40' (blank)	Intermediate portion of message
X'F2' (2)	Last portion of message
X'F3' (3)	An entire message

SYNAD=address

Alternate Source: PP.
Function: Specifies the address of a routine to be given control when logical output errors occur.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: For more information on this routine, see *Application Program Error Exit* in this chapter.

RECFM= { **F** / **V[B]** / **U** }

Alternate Source: DD.
Function: Specifies the format of the work units in this output data set.
Default: RECFM=U
Format: F, V, VB, or U.
Notes: V specifies that the work units are variable in length. For BSAM and QSAM, each work unit is prefaced in the work area by a standard SAM variable-length record prefix of four bytes (the contents of which are in hexadecimal format). The length of the work unit must be provided by setting up the prefix before issuing a PUT or WRITE macro.

If RECFM=VB, the records are treated as blocked, although only one work unit is transferred to the MCP per PUT or WRITE macro. The variable-length record work area includes a blocked work area prefix of eight bytes if MACRF=W is specified, and four bytes if otherwise.

U specifies undefined-length work units. TCAM, like SAM, provides no prefix. 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 LRECL= field of the DCB prior to each PUT or WRITE, unless it is specified by the length operand of the WRITE macro.

F specifies fixed-length work units. Prior to the PUT or WRITE, 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 LRECL= field of the output DCB.

EXLST=address

Alternate Source: PP.
Function: Specifies the address of the problem-program exit list.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: The description of this operand is the same as that provided above for the EXLST= operand of the input DCB macro.

BUFL=integer

Alternate Source: DD.
Function: Specifies the length in bytes of the MCP buffers that are to receive messages coming from this application program.
Default: None. Specification optional.

Format: Unframed decimal integer greater than 35.
Maximum: 65535
Notes: If this operand is omitted, the value specified in the BUFSIZE= operand of the PCB macro in the MCP is the value used.

DD Statements for the Input and Output Data Sets

At application-program execution time, one DD statement must be provided for each DCB. The DD statement has the following format:

```
//ddname DD QNAME=procname
```

ddname

Is the symbolic name of the DD statement, and must be the same as the name specified in the DDNAME= operand of the input or output DCB macro.

procname

Is the name of the process entry in the terminal table to which this entry refers. This name is assigned by the TPROCESS macro creating the entry. The destination queue may be changed at execution time by specifying a different value for the QNAME= parameter.

NOTE: The following DCB operands may be omitted from the input or output DCB macro and coded as parameters of the DD statement when the operand's functions are to be provided by an alternate source. These operands are explained in the discussion of the input and output DCB macros. More than one operand can be specified in one DCB= parameter; multiple operands should be separated by commas.

```
[,DCB=([BLKSIZE=integer]
      [,BUFL=integer] [,LRECL=integer]
      [,OPTCD=[W][U][C]]
      [,RECFM={U    } ])
              {V[B] }
              {F    }
```

PCB Macro Instruction

The PCB macro:

- Provides a control block in the MCP to interface with an application program,
- Is required for each application program running with the MCP,
- Is coded in the MCP, not the application program.

The PCB macro generates a named control block, known as a *process control block* (PCB). A process control block is a vehicle to provide information needed to communicate between the MCP and an application program. One and only one PCB macro is required for each active application program.

The PCB macro has the following format:

Name	*Operation*	*Operands*
pcbname	PCB	MH=mhname, BUFSIZE=size [,BUFIN={number}] [,BUFOUT={number}] { 2 } { 2 } [,RESERVE=(integer1,integer2)]

pcbname

Function: Specifies the name of the macro and the name of the process control block generated by the macro referred to in the TPROCESS macro.
Default: None. This name is required.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

MH=mhname

Function: Specifies the symbolic address of the Message Handler for the application program represented by this macro.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols and be identical to the name specified in the name field of a STARTMH macro in the Message Handler.

BUFSIZE=integer

Function: Specifies the size of the buffers to be assigned to handle messages for the associated application program.
Default: None. This operand is required.
Format: Unframed decimal integer greater than 35.
Maximum: 65535
Notes: This value may be overriden by specifying the BUFL= operand of the input or output DCB for the application program.

BUFIN={**number** / **2**}

Function: Specifies the initial number of buffers requested into which the data in the user's PUT/WRITE work area will be emptied.
Default: BUFIN=2
Format: Unframed decimal integer greater than 1.
Maximum: 15
Notes: The optimum number specifies enough buffers to contain the entire work area.

BUFOUT={**number** / **2**}

Function: Specifies the initial number of buffers that may be filled in anticipation of a GET or READ.
Default: BUFOUT=2
Format: Unframed decimal integer greater than 1.
Maximum: 25
Notes: Used as a read-ahead queue for a process entry.

RESERVE=(integer1,integer2)

Function: Specifies the number of bytes to be reserved in buffers.
Default: None. Specification optional.
Format: Unframed decimal integers.
Maximum: 255 for each.
Notes: *integer1* specifies the number of bytes to be reserved in the buffer receiving the first incoming segment of each message entered by an application program; the space is reserved for insertion of data by DATETIME and SEQUENCE functional MH macros.

integer2 specifies the number of bytes to be reserved in all buffers except the first for insertion of characters by the DATETIME macro. *integer2* is relevant only in a multiple-buffer header situation when the DATETIME macro is to insert data in a portion of the header that is not in the first buffer (see the description of the DATETIME macro for an example of when it might be desirable to execute DATETIME on a portion of the header not located in the first segment).

Data may be inserted in either an incoming or an outgoing message header, but space must be reserved in the incoming header. On the outgoing side, reserved space is retained for the first buffer only; thus, DATETIME and SEQUENCE macros, if specified in an out-header subgroup, operate on the first segment of the message.

No space need be reserved for data inserted by a MSGEDIT functional MH macro.

The *Scan Pointer* section of the chapter *Designing a Message Handler* describes how TCAM handles reserve bytes. Each buffer containing header data should be large enough to accommodate the segment itself plus any data that may be inserted by DATETIME and SEQUENCE macros. If a buffer containing header data does not have a sufficient number of bytes reserved in it to accommodate data inserted by a DATETIME or SEQUENCE macro, the macro does not execute and control passes to the next instruction in the MH. Unused reserve bytes are not sent out with an outgoing message segment when it is sent to its destination.

Defining Buffers for the Application Program

Like messages being transferred between the MCP and a remote station, messages being transferred between the application program work area and the MCP reside in buffers. The buffers for transferring data to and from the application program are ordinary TCAM buffers, as described in *Defining Buffers.* That chapter should be read and understood by the programmer responsible for defining the application-program buffers, as the structural description and most of the design considerations in that chapter are also applicable to application-program buffers.

Buffers used to transfer data between an application program and an MCP differ from those assigned to a line in two respects;

- The way in which they are defined;
- The manner in which they are allocated.

The next section describes application-program buffer definition. The following section describes the allocation scheme for application-program buffers as part of a discussion of application-program buffer design considerations.

Defining Application-Program Buffers

A buffer-definition checklist for the application-program buffers follows. Guidelines for coding many of the operands shown are given in the next section.

Macro	*Operand*	*Description of Function and Comments*
INTRO	KEYLEN=integer	Specifies the length in bytes of a buffer unit; all buffers in the TCAM system are constructed of units of this size. (Considerations for coding this operand are given in the chapter *Defining Buffers.*) *integer* must be between 31 and 255 inclusive.
PCB	BUFSIZE=integer	Specifies the length in bytes of the buffers to transfer message segments between the process queues for the application program and an application-program work area. May be overriden for a single input or output data set by the BUFL= operands of the input or output DCB macro for that data set. *integer* must be between 31 and 65535 inclusive.
	[BUFOUT={integer / 2}]	Specifies the maximum number of application-program buffers that may be filled from the destination queue, processed by the outgoing group of the application-program MH, and placed on the read-ahead queue in main storage in anticipation of a GET or READ macro. *integer* must be at least 2 (TCAM uses one buffer internally) and may be no greater than 25.
	[BUFIN={integer / 2}]	Specifies the initial number of buffers to be allocated to receive data being transferred by a PUT or WRITE macro from the application-program work area to the MCP. *integer* may be between 2 and 15 inclusive.
Input DCB	[BUFL=integer]	Specifies the length in bytes of the buffers to be used to transfer message segments from the MCP to the application program; overrides the value specified by the BUFSIZE= operand of the PCB macro. *integer* must be between 31 and 65535 inclusive.
Output DCB	[BUFL=integer]	Specifies the length in bytes of the buffers to be used to transfer messages segments from the application program to the MCP; overrides the value specified by the BUFSIZE= operand of the PCB macro. *integer* must be between 31 and 65535 inclusive.

Application-Program Buffer Design Considerations

The user assigns a maximum number of buffers that can be used at one time to handle messages being transferred from MCP process queues to the application-program work area via the BUFOUT= operand of the PCB macro. These buffers are used to construct the *read-ahead queue*, a temporary queue in main storage on which messages are held in anticipation of a GET or READ. The read-ahead queue is discussed in *Message Flow to an Application Program* in this chapter. TCAM constructs one read-ahead queue for each process queue associated with an opened input data set.

The maximum capacity of a read-ahead queue is two messages. Buffers are allocated to this queue dynamically, but the queue never contains more than the number of buffers needed to handle two messages. If the user specifies (with the BUFOUT= operand of the PCB macro) a number of buffers less than that needed to contain two entire messages on the read-ahead queue, less main storage is tied up by being assigned to the read-ahead queue, but more time is required to transfer messages to the application program.

The following formula for calculating the BUFOUT= operand of the PCB macro provides a read-ahead queue always capable of containing two complete messages; by specifying a queue of this size, the user minimizes delay in transferring messages to the application program:

I=2X+1

Here *I* represents the integer to be coded 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 sole type of queuing used for process queues, the optimum number of buffers specified by BUFOUT= is reduced; in this case, one need specify only enough buffers to handle the largest work unit to be sent to the application program for optimal-performance read-ahead queues.

The BUFIN= operand of the PCB macro specifies the initial number of buffers to 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 application-program process entry that may be referred to by PUT or WRITE macros, the number of buffers specified by BUFIN= is allocated to each.) Buffers assigned to receive data from the application program are deallocated and sent through the incoming group of the application-program message handler as they are filled.

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 allocation takes time; to optimize performance, a sufficient number of buffers should be assigned initially to handle the entire work unit.

The size of the application-program buffers is specified by the BUFSIZE= operand of the PCB macro. This size may be overridden for buffers handling data being transferred to the application program by the BUFL= operand of the input DCB macro, and for buffers handling data being transferred from the application program by the BUFL= operand of the output DCB macro.

Buffer size considerations given in the chapter *Defining Buffers* are relevant to application-program buffers (considerations in that chapter that deal with program-controlled interruptions (PCI) are an exception).

Buffers are sent through the incoming group of the application-program MH as soon as they are filled. If a buffer is not 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 upon whether message- or record-processing is specified. (Position fields are discussed in *Defining Optional Fields in the Work Area* in this chapter. Message- and record-processing are described in *Specifying Application-Program Work Units.*)

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 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 fitted by a subsequent PUT or WRITE (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 space is wasted, as explained above. If the position field indicates that the current work unit is the first or an intermediate unit in a multi-unit 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 above.

When the buffer sizes specified for the origin and the destination of a message are different, data movement occurs because prefixes must be added or deleted when the message is placed in the buffers for the destination (this is discussed in the chapter *Defining Buffers*). Because data movement takes time, the buffer size for line buffers handling messages being sent to or from an application program should be the same as the buffer size for the application-program buffers wherever possible. By overriding the buffer size specified by the BUFSIZE= operand of the PCB, the BUFL= operand of the input and output DCB macros may be used to tailor application-program buffer sizes to buffer sizes for particular origin or destination stations.

For example, if line buffers for all stations that could enter and accept messages processed by a particular application program were either 116 bytes long or 232 bytes long, the user could define two input and output data sets (each with its own GET/READ and PUT/WRITE process entries), one for each buffer length. He could direct all incoming messages for the application program that were entered by stations using 116-byte buffers to one process queue, and all incoming messages for the application program that were entered by stations using 232-byte buffers to the other process queue. If he 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 at the computer, no data transfer would be necessary when the data was read from the destination queue into application-program buffer.

When transferring responses from the application program, the user would name the PUT/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 destination station. In the output DCB for the 232-byte-buffer output data set, he would specify BUFL=232. Again, no data transfer would be necessary when messages were read from the destination queues into the line buffers for the destination station if this scheme were followed.

Activating and Deactivating the Application-Program Interface

Activation and deactivation of the interface between an application program and the MCP is handled by OPEN, CLOSE, and MCPCLOSE macro instructions. The OPEN and CLOSE macros for TCAM-compatible application programs are used and coded in the same way as OPEN and CLOSE macros coded for application programs in a non-teleprocessing environment and described in the *Supervisor and Data Management Macro Instructions* publication. List and execute forms may be coded for OPEN and CLOSE. The user may code options for the OPEN and CLOSE macros shown in *Supervisor and Data Management Macro Instructions* to run his application program in a non-teleprocessing environment for debugging purposes; when the program is run in a TCAM environment, the option fields are ignored. More than one data set may be opened or closed with the same application-program OPEN or CLOSE macros. The OPEN, CLOSE, and MCPCLOSE macros are described in this section. Deactivation of the application program is discussed in the chapter *Activating and Deactivating the Message Control Program.*

Open Macro Instruction for the Application Program

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.

Initialization and activation of the interface to the MCP is accomplished by issuing one or more OPEN macros to open the data sets represented by the input and output DCB macros.

One input DCB macro must be coded for each process queue for an application program (i.e.,each queue for which messages can be obtained by GET or READ macros). One output DCB macro must be coded for each process entry that can be referred to by a PUT or WRITE macro when a work unit is being transferred from the application program to the MCP.

The open routines in TCAM activate the interface between the MCP and the application programs. No TCAM macro instructions in the application program may be successfully executed before the DCB for the message queues data set has been opened in the MCP or after it has been closed (if disk queuing is used), or before the input and output data sets are opened or after they are closed. After the message queues data sets on disk and application-program data sets have been opened, transfer of data to and from the application program can commence.

The operand field of the OPEN macro consists of one or more positional operands, followed by a single keyword operand. Each positional operand consists of the name of the data control block for the data set being opened (the name of the block is the name of the DCB macro that created it). A comma is coded between names. The optional keyword operand at the end permits the list and the execute form of the macro to be specified.

The OPEN macro for the application program has the following format:

Name	*Operation*	*Operand*
[symbol]	OPEN	(dcbname,,...)[, MF={L / (E,listname)}]

symbol

Function: Specifies the name of the macro.
Default: None. If MF=L is coded, this name is required. Otherwise specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).
Notes: If MF=L is specified, this name becomes the name of the parameter list generated by this macro.

(dcbname,,...)

Function: Specifies the name of the data control block and is identical to the name specified in the symbol field of the DCB macro for the data set being opened.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: Register notation may be used, in which case the specified register (2 through 12) should contain the address of the data control block for the data set being opened. The specified register number must be enclosed in parentheses. If more than one *dcbname* is specified, they must be separated by double commas.

MF={L / (E,listname)}

Function: Specifies that a list is to be created, or that a previously created list is to be opened.
Default: None. Specification optional.
Format: *listname* must conform to the rules for assembler language symbols.
Notes: MF=L causes creation of a parameter list based on the OPEN operands. No executable code is generated. The user must specify this form of the OPEN among his program constants. The parameters in the list are not used until the problem program issues an OPEN (or CLOSE) macro with an MF= (E,listname) operand that refers to the list. The name specified in the name field becomes the name assigned to the parameter list.

MF= (E,listname) causes execution of the OPEN routine, 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.

CLOSE Macro Instruction for the Application Program

The CLOSE macro:

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

The CLOSE macro has the following format:

Name	*Operation*	*Operand*
[symbol]	CLOSE	(dcbname,,...)[MF={L / (E,listname)}]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

(dcbname,,...)

Function: Specifies the name of the data control block(s) for the data set(s) being closed.
Default: None. This operand is required.
Format: Framing parentheses must be coded. Each *dcbname* must conform to the rules for assembler language symbols, and must be the same as the name of the DCB macro creating the control block.
Notes: All application-program data sets can be closed with one CLOSE macro by including the names of their data control blocks as operands.

If register notation is used, the register number must be enclosed in parentheses, and addresses of the data control blocks must previously have been loaded into the registers specified.

If more than one data set is being closed, the names must be separated by double commas.

MF={L / (E,listname)}

Function: Specifies list or execute form of the macro.
Notes: See the OS publication *Supervisor and Data Management Macro Instructions* for the definition and use of this operand.

System ABEND issues CLOSE macros for all opened DCBs within a task when it abends. Open DCBs are found by means of a scan of the DEB chain contained in the TCB of the task to be terminated abnormally.

MCPCLOSE Macro Instruction

The MCPCLOSE macro:

- Initiates closedown of the telecommunications system,
- Is optional in an application program.

MCPCLOSE may be issued in an application program to initiate system closedown. At the time MCPCLOSE is issued in a user-written termination routine, all data sets in the application program should be closed (if MCPCLOSE detects an open data set in any application program, it issues a WTO message and places the MCP in a wait state until all data sets are closed). Following successful execution of MCPCLOSE, control passes to a user-specified routine that deactivates the MCP. For more information on the use of MCPCLOSE, see *Deactivation* in *Activating and Deactivating the Message Control Program.*

Only one MCPCLOSE macro is needed to close down the entire system. The closedown functions of the macro are also available through use of the SYSCLOSE operator command.

One of the following return codes is returned to the application program in register 15 after the MCPCLOSE macro is issued:

Code	Meaning
X'00000000' X'0000000C' X'00000014'	The MCPCLOSE macro executed successfully. TCAM is not in the system. Either a) An invalid protection password is specified in the PASSWRD= operand, or b) The PASSWRD= operand is not specified and is needed because the INTRO macro's PASSWRD= operand specifies a protection password.

The MCPCLOSE macro has the following format:

Name	Operation	Operand
[symbol]	MCPCLOSE	{QUICK}[,PASSWRD=chars] {FLUSH}

symbol

Function: Specifies the name of the macro instruction.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

{QUICK}
{FLUSH}

Function: Specifies the type of closedown required.
Default: FLUSH
Format: QUICK or FLUSH.
Notes: QUICK specifies that message traffic is to cease upon completion of any message currently in progress. Messages queued for the destinations are not transmitted.

FLUSH specifies that input message traffic is to cease upon completion of the message currently in progress. All messages queued for the destinations are then transmitted.

PASSWRD=chars

Function: Specifies the protection password that enables only qualified application programs to issue the macro.
Default: None. If the PASSWRD= operand is specified on the INTRO macro in the MCP, this operand is required. Otherwise, specification optional.
Format: One to eight unframed nonblank characters.
Notes: If the character string specified in this operand does not match that specified in the INTRO macro, the MCPCLOSE macro is ignored and a X'14' return code is set in register 15.

Transferring Data Between an MCP and an Application Program

TCAM provides the application program user with facilities for obtaining messages from the MCP, and for returning response messages to the MCP.

Although the messages are received from (or sent to) remote stations over communication lines, the application programmer uses data-transfer macros similar to those of the Queued Sequential Access Method (GET and PUT) or the Basic Sequential Access Method (READ, WRITE, and CHECK) of OS/360. A TCAM Message Control Program performs device-dependent input/output operations for the application program.

Since the macros used by TCAM for transfer of data between an application program and an MCP are patterned after those of BSAM and QSAM, the TCAM application programmer is expected to be familiar with these access methods, which are explained in the *Supervisor Services, Data Management Services,* and *Supervisor and Data Management Macro Instructions* publications.

The amount of data transferred from the MCP to an application program by a single GET or READ macro, or transferred from an application program to the MCP by a single PUT or WRITE macro, is called a *work unit.* The work unit is processed in an

application-program *work area.* A work unit may be an entire message, or a portion of a message (which may or may not be a record). A *message* is a unit of data received from or sent to a station and terminated by an EOT or ETX line-control character or, if the CONV= operand of the STARTMH macro is coded CONV=YES, by an ETX or EOB line-control character. (Line-control characters may be deleted by the MCP, but TCAM places the length of each message segment in the buffer prefix for that segment, and can determine the message length by adding the contents of the prefix fields.)

A *record* is a logical unit of data whose length is defined by operands of the input or output DCB macro and delimiting characters in the message. In TCAM, each record is transferred to and from a remote station as part of a message, but the size of the record need not coincide with the size of the message; one message may contain many records. After an incoming message is placed on a process queue for the application program, the user obtains the records in it one at a time, with one record being passed between the MCP and the application-program for each GET or READ macro directed to the process queue. Similarly, a record may be sent to the MCP from a work area whenever a PUT or WRITE macro naming the work area is issued in the application program.

Just because a work unit is not an entire message does not mean that it is a record. Message processing or record processing is indicated by the OPTCD= operand of the input and output DCB macros. If message processing is specified, but the entire message does not fit into the work area, TCAM provides the capability of processing a portion of the message in the work area, then bringing in the next portion and processing it, until the entire message has been processed. The portions of the message processed in this way are not considered to be records, since message processing rather than record processing was specified; TCAM handles records and other message portions differently, as shown below in the discussion of work units and work areas. These differences may be summarized as follows:

- An incoming record cannot overflow the work area, whereas an incoming message can.
- An incoming record may be delimited by a delimiting character specified by the RECDEL= operand of the TPROCESS macro; when message processing is specified in the input DCB macro, such delimiters are ignored.
- If neither a delimiting character nor end-of-message is encountered in a record by the time the work area is full, the size of the record is assumed to be the size of the work area. When message processing is specified, a work-area overflow condition is assumed to exist if the work area fills before the entire message is read in; in this case, the user specifies by an input DCB operand, whether he wants to process the message piece by piece or go to an error routine.
- If a delimiting character is specified by the RECDEL= operand of the TPROCESS macro named in a PUT or WRITE macro, TCAM places the character at the end of each outgoing record. If message processing is specified, TCAM places no delimiting character at the end of outgoing messages or pieces of messages.

The next three sections of this chapter discuss in detail the application-program work-area, work-unit, and data-transfer macros.

Defining the Application-Program Work Area

Work units obtained by a GET or READ macro are transferred from the MCP to a *work area* defined by the user when he codes his application program. The work areas for TCAM-compatible application programs are similar to those for programs using the Basic or Queued Sequential Access Method.

Static Work-Area Definition

A work area may be defined in one of two ways. It may be defined at application-program assembly time by a DC or DS assembler instruction issued in the application program. The label of the instruction becomes the name of the work area, and is coded in the GET, PUT, READ or WRITE instructions that move data to and from the work area. The size of the work area must be specified in the BLKSIZE= operand of the input DCB macro associated with the data set whose contents are being transferred to or from the work area.

When a work area is defined in this way, move processing mode should be specified by coding M in the MACRF= operand of the DCB macros 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.

Dynamic Work-Area Definition

A work area may be defined dynamically at application-program execution time, if GET or PUT macros are to gain access to it. If the user specifies locate mode by coding L in the MACRF= operand of his 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 area of addressability as the application program, and to move a work unit of data into this work area. The length of the work area is that specified by the BUFSIZE= operand of the input DCB macro referred to by the GET macro. The work area's address is returned in register 1, and is saved by TCAM. The second and subsequent executions of GET macros referring to the DCB move data into this work area.

If locate mode is specified by coding L in the MACRF= operand of the output DCB macro, execution of the first PUT macro referring to the opened data set causes TCAM to dynamically obtain a work area (by a GETMAIN macro) in the same area of addressability as the application program. The address of this work area is returned in register 1. This address should be saved by the user and placed in register 1 before each PUT after the first is issued. The length of this work area is specified by the BLKSIZE= operand of the output DCB referred to by the PUT macro. The user must move his data into the work area before executing another PUT referring to this DCB. Execution of subsequent PUT macros referring to this DCB moves the data from this same work area to the MCP buffers.

Moving Data between Input and Output Work Areas

In some user applications, a work unit is transferred from the MCP to the application program by a GET or READ, processed by the application program, and then returned to the MCP by a PUT or WRITE. If move mode is specified in the input and output DCB macros for the input and output data sets through which the work unit proceeds, then the GET/READ and PUT/WRITE macros may refer to the same work area, so that the user need not move his data from an input to an output work area.

If locate mode is specified in the input or output DCB macro, and move mode is specified in the DCB macro for the other data set through which the work unit passes, then the user can still get by with one work area, because TCAM permits specification of a register containing the address of the work area when GET or PUT is coded.

If locate mode is specified for both the input and the output DCB macro, then two work areas will be present, and the work unit must be transferred from one to the other.

Defining Optional Fields in the Work Area

The following operands of the input and output DCB macros cause TCAM to create optional fields in the front part of the work area and fill them with data (input DCB macro) or to examine these fields (output DCB macro):

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

If none of these operands are coded, TCAM starts with the first byte of the work area when filling or emptying it.

The contents of the optional fields are not moved out of the work area with the message or record being processed.

Origin and Destination Fields

If W is coded in the OPTCD= operand of the DCB macro of the input data set for this work unit, eight bytes of the work area are reserved for the name of the source of the message. When the message comes into the work area, TCAM places the EBCDIC name of the source (as specified in the terminal table) into these eight bytes. The name is left-adjusted, and the field is padded to the right with blanks if the name is shorter than eight bytes.

If TCAM cannot determine the origin of a message, the field is filled in with eight character blanks. TCAM usually knows the origin of a message. TCAM does *not* know the origin when a switched station with no ID sequence calls in and fails to identify itself by having a valid origin field in the message header checked by an ORIGIN macro. If the switched station is assigned an ID sequence that is not unique, an incorrect name may be placed in the field. (See the discussion of the ORIGIN macro for more information on switched stations with no ID sequences or non-unique ID sequences.)

The eight-byte origin field immediately precedes the work unit in the work area, and follows the other two optional fields if either or both of the other fields are present. Figure 27 shows where the origin field goes in the work area.

If W is coded in the OPTCD= operand of the DCB macro of the output data set for this work unit, when a PUT or WRITE macro is issued to move a work unit from this work area to the MCP, TCAM looks in an eight-byte field in the work area for the name of the destination of the message. The name should be 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 in the inheader subgroup of the Message Handler for an application program, the message is sent to the destination specified in the eight-byte field (see the description of the FORWARD macro).

TCAM assumes that the eight-byte destination field immediately precedes the work unit in the work area (if W is coded in the OPTCD= operand); Figure 27 shows where TCAM looks for the destination field. Only the work unit, and not the contents of the destination field, is transferred to the MCP when a PUT or WRITE macro is executed.

The user with an inquiry-response application may wish to refer to the same work area with his GET/READ and PUT/WRITE macros; if he codes W in the OPTCD= operands of his input and output DCB macros, TCAM places the origin in the eight-byte field when the inquiry message is 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 the destination. If a FORWARD macro with the DEST= operand coded DEST=PUT is coded in the inheader subgroup for the application program, the response message will go back to the originating terminal.

Position Field

If C is coded in the OPTCD= operand of an input or output DCB macro, a one-byte field is reserved in the work area associated with the DCB (if locate mode is specified in the DCB macro),or named by the GET/READ or PUT/WRITE macro transferring data to or from an input or output data set. This field is useful when messages sent to the application program are larger than the application-program work area that is to receive them (e.g., when logical records or other message portions, rather than entire messages, are processed by the application programs). This byte contains a code indicating whether the work unit being processed is the first portion of a message, an intermediate portion, the last portion, or an entire message (these codes are given in the discussion of the OPTCD= operand of the input and output DCB macros).

If C is specified in the OPTCD= operand of the input DCB macro containing the work unit to be moved into the work area, TCAM fills in the position field with a code indicating whether this work unit is the first, intermediate, or last portion of a message, or an entire message.

If C is specified in the OPTCD= operand of the output DCB macro for the work unit, the application programmer must ensure that the position field contains the appropriate code to describe his work unit. TCAM checks this field and uses it to account for message portions being transferred to the MCP. The user must not interleave segments from

different messages. If the operand is omitted from the output DCB macro, TCAM must make one of two assumptions, depending upon whether record processing or message processing is specified in the OPTCD= operand of the output DCB macro (message processing and record processing are described in the next section).

- If message processing is specified, the end of the work unit is assumed to be the end of the message–i.e., TCAM assumes that one work unit equals one message.
- If record processing is specified, TCAM assumes that all work units being sent to the process entry associated with this output DCB, from the time the output data set is opened until the time it is closed, are part of the same message–i.e., 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 to the left of the eight-byte origin or destination field. If no origin or destination field is present, the position field is located immediately to the left of the first byte of message data in the work area. Figure 27 shows the location of the position field in the work area.

SAM Prefix

If V or VB is coded in the RECFM= operand of the input or output DCB macro, a prefix field is assumed to be present in the work area containing the message received from or sent to the data set represented by the DCB. This prefix is useful when TCAM/SAM compatibility–the ability to run application programs in a non-teleprocessing environment using SAM data sets, and then run the same program in a TCAM environment without reassembling–is desired (see *TCAM/SAM Compatibility* in this chapter). In addition, TCAM needs a SAM prefix when variable-format work units are specified in the output DCB macro (such work units are discussed in *Work-Unit Formats* in this chapter).

The SAM prefix, if present, occupies the first four or eight bytes of the work area, as shown in Figure 27.

If RECFM=V is coded in the input DCB macro, TCAM places a four-byte prefix into the work area receiving a work unit from the input data set for which the DCB macro was coded. The first two bytes of the prefix 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 each contain binary zeros.

If RECFM=VB is coded in the input DCB macro, 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. 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 execution.

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

If RECFM=VB is coded in the output DCB macro, 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.

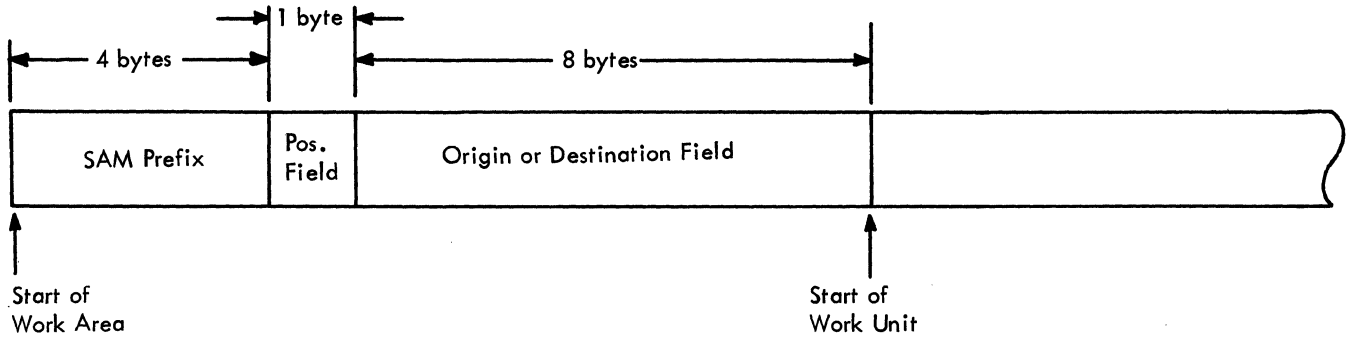


Figure 27. Relative Positions of Optional Fields in the Work Area

Specifying Application-Program Work Units

The way in which TCAM decides how long a work unit is and how to handle it depends upon two factors:

- The *format* of the work unit (fixed, variable, or undefined);
- The *type* of work unit (message or record).

The user specifies the format and type of work units his application program is to process by coding operands of the input and output DCB macros for the application program. These operands indicate whether the work unit is a message or a record, and whether it is always the same length or may vary in length from message to message or from record to record.

If messages or records sent to an application program may vary in length, user code in the application program will want to know the length of the work unit currently being processed. TCAM counts the number of bytes in the incoming work unit, adds the number of bytes that must be reserved in the work area for optional fields, and places the total either in a special field in the work area or in a field in the input DCB (depending upon which field the user specifies in an input DCB operand). User code may then inspect this field to determine the length of the work unit being processed.

On the output side of the application program, TCAM must know the length of messages or records whose lengths may vary, before these work units can be transferred to the MCP. 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 special field in the work area or output DCB before issuing a PUT or WRITE macro to transfer the work unit.

The next two sections of this chapter discuss the effects of work-unit format and type upon the way in which TCAM transfers the work unit to and from the application program. Figure 28, at the end of the second section, summarizes much of this discussion.

Work-Unit Formats

Work units that always have the same length are said to have a fixed format, while work units that may vary in length may have either a variable format or an undefined format, depending upon the location of the field in which their length is stored (for incoming work units) or examined (for outgoing work units) by TCAM.

A *fixed-format work unit* is one whose length is defined by the number of bytes coded in the LRECL= operand of the input and output DCB macros. A *variable-format work unit* is one whose length (plus the length of any optional fields in the work area) is stored in the SAM prefix in the work area (see *SAM Prefix* in this chapter). An *undefined-format work unit* is one whose length (plus the length of any optional fields in the work area) is stored in a field in the input or output DCB. TCAM counts the number of bytes in the incoming work unit for both variable- and undefined-format 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 where the count is stored by TCAM.

The tables at the end of this section summarize the specification and characteristics of the various work-unit formats.

The user specifies the kind of work units his application program expects to accept by the RECFM= operand of the input DCB macro. If he codes RECFM=F, then TCAM knows that his application program is set up to process fixed-length work units, and looks for the length of these units in the LRECL field of the input DCB. If he codes RECFM=V, then TCAM keeps track of the length of each incoming work unit and stores in the SAM prefix the sum of this length plus the length of any optional fields in the work area. If he codes RECFM=U, then TCAM keeps track of the length of each incoming work unit and stores the sum of this length plus the length of any optional fields in the work area in the LRECL field in the input DCB.

For work units being transferred from an application-program to the MCP, a similar setup prevails. The user tells TCAM, by the RECFM= operand of the output DCB macro, where to look for the length of the work unit being sent back to the MCP by each PUT or WRITE macro. If the user specified RECFM=F, TCAM looks for the length of the work unit in the LRECL field of the output DCB. If the user specifies RECFM=V, TCAM looks for the sum of the length of the work unit plus the length of any optional fields in the work area in the length field of the SAM prefix in the work area. If the user specifies RECFM=U, TCAM looks for 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 output DCB if a PUT or WRITE is being issued. It is up to the user to ensure that the field TCAM examines contains the correct length; the technique for modifying a DCB field is described in *Data Management Services.* If the WRITE macro is used with the *length* operand, the length specified in the WRITE macro is used.

The tables below summarize the TCAM work-unit formats and illustrate how they are specified by operands of the input and output DCB macros.

The delimiter mentioned in the discussions of variable and undefined 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 specified by the RECDEL= operand of the TPROCESS macro creating the queue accessed by the GET or READ.

Work-Unit Formats—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 attempts to bring in the number of bytes specified by the LRECL= operand of the input DCB macro.
Variable	RECFM=V[B]	Incoming work units 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, and then places the sum of the length of the work unit plus the length of any optional fields in the work area in the SAM prefix, which precedes the work unit in the work area.
Undefined	RECFM=U	Incoming work units 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, and then places in the LRECL field in the input DCB the sum of the length of the work unit plus the length of any optional fields in the work area.

Work-unit Formats–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.
Variable	RECFM=V[B]	When a PUT or WRITE macro referring to this DCB is executed, TCAM determines the length of the work unit to be moved by looking in the SAM prefix preceding the work unit in the work area (and subtracting the length of any optional fields in the work area).
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, TCAM subtracts the length of any optional fields in the work area from the value found.) If the WRITE macro with the *length* operand referring to this DCB is executed, TCAM uses the length specified in the WRITE macro as the length of the work unit to be moved.

Work Unit Types

An application program may be set up to handle messages or records. A work unit may be a message or a portion of a message; a work unit that is a portion of a message may be, but need not be, a record.

The terms *message* and *record* are defined above, in the section *Transferring Data Between an MCP and an Application Program;* differences in the manner in which TCAM handles records and the manner in which it handles other message portions that are not records are also summarized under this heading. The table at the end of this section gives a more detailed contrast between message and record processing by TCAM.

The user specifies that he has set up his application program to handle messages by coding U in the OPTCD= operand of the input DCB macro. If U is not coded, TCAM assumes that the incoming work unit is a record.

Processing the Message as a Work Unit: If U is coded in the OPTCD= operand of the input DCB macro, TCAM attempts to read in an entire message when a GET or a READ macro is executed. If the work area is large enough to accommodate the entire message, TCAM reads in data up to and including the EOT or ETX line-control character unless conversational mode is specified in the STARTMH macro, in which case TCAM reads in a block of data (i.e., that amount of data delimited by an EOB or ETB character when received by the computer). If LC=OUT is specified in the STARTMH macro associated with the line group DCB macro, the EOB or ETB line-control character is removed when the message comes into the incoming group of the MCP, but the EOT or ETX line-control character is *not* removed.

If the entire message does not fit into the designated work area, TCAM performs one of three actions, depending upon how operands of the input DCB macro are coded:

1. If a position field is specified in the OPTCD= operand, the portion of the message that did not fit into the work unit is obtained by the next GET or READ macro executed (the position field is discussed in *Optional Fields in the Work Area* in this chapter);
2. If no position field is specified but SYNAD= is coded, TCAM gives control to the routine specified by SYNAD= (this routine is discussed in *Application-Program Error-Handling Facilities* in this chapter);
3. If neither a position field nor a SYNAD exit is specified, TCAM places a return code of X'00000008' in register 15. This return code indicates an error condition, and the user should terminate the application program and correct the error.

If TCAM performs the first of these three actions, the application program may process the first portion of the message and issue a PUT or WRITE macro to return it to the MCP before issuing a GET or READ to bring in the rest of the message.

To determine whether an incoming message fits into the work area, TCAM must first know what the length of the work area is. For fixed-format messages, TCAM assumes that the length of the work area is equal to the number of bytes specified in the LRECL= operand of the input DCB macro. For variable- and undefined-format 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. 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 not the message that caused the overflow; this message is thrown away by TCAM.

To prevent work-area overflow, the CUTOFF macro can be coded in the inbuffer subgroup of the MH; this macro checks the length of incoming messages and permits cancellation of messages that would be too long for the work area.

If U is specified in the OPTCD= operand of the output DCB macro, TCAM assumes message processing on the output side. 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 portion 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 as part of the same message, until the contents of the position field indicate 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 located in the work area.

Depending upon the format of the work unit (i.e., whether it is fixed, variable, or undefined), TCAM looks in the SAM prefix 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, after allowing for optional fields in the work area. (See *Work-unit Formats* in this chapter for information on the exact location of the field containing the message length.)

Processing the Record as a Work Unit: If U is not coded in the OPTCD= operand of the input DCB macro, TCAM treats the incoming work unit as a record, rather than as a message or a message portion that is not a record.

If the user specifies that the input to his application program is to be fixed-format records (by coding RECFM=F in the input DCB macro), TCAM assumes that each incoming record is equal in length to the number of bytes specified in the LRECL= operand of the input DCB macro (minus the length of any optional fields in the work area) and moves in this number of bytes each time that a GET or READ macro is executed for this input data set. The last record in a message may be shorter than the number of bytes specified by LRECL=, in which case TCAM brings in the actual number of bytes in this record.

If fixed-format 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 is executed

TCAM transfers to the MCP a record equal in length to the number of bytes specified in the LRECL= operand of the output DCB macro (after making allowance for the length of optional fields in the work area).

If the user specifies that the input to his application program is to be variable- or undefined-length records (by coding V or U, respectively, in his input DCB macro), TCAM determines the length of incoming records according to the following principles:

1. If a delimiting character (specified by the RECDEL= operand of the TPROCESS macro creating the destination queue accessed 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.
2. 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.
3. 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 field of the input DCB.

When record processing is specified in the DCB macro for the output data set, TCAM sends out a single record with each PUT or WRITE. The size of the record is indicated in the SAM prefix for variable-format records and in the LRECL field of the output DCB or in the WRITE macro for undefined-format records transferred by a PUT or WRITE.

The table below and Figure 28 summarize many points discussed in this section and in the one immediately preceding it.

Differences between Message and Record Processing

	MESSAGE PROCESSING	RECORD PROCESSING
On Input:	• When GET/READ is issued, TCAM brings in data until either the end of the message is encountered or the work area is filled.	• When GET/READ is issued, TCAM brings in data until 1) the delimiting character specified by the TPROCESS macro referred to by the GET or READ is encountered, or 2) the end of the message is encountered, or 3) the work area is filled. (Delimiting character is ignored for fixed-format records).
	If the work area has been filled and the end of the message was not reached, TCAM either brings in the rest of the message with the next GET or READ (if a position field is present in the work area), 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.

Differences between Message and Record Processing (Continued)

	MESSAGE PROCESSING	RECORD PROCESSING
On Output:	• Whenever PUT/WRITE is executed, TCAM transfers one work unit of data from the application-program work area to the MCP.	• Same as for messages – one work unit (record) transferred per PUT or WRITE.
	• The RECDEL= operand of the output TPROCESS macro is ignored.	• The delimiting character specified by the RECDEL= operand of the output TPROCESS is placed at the end of each outgoing undefined-format record and each outgoing variable-format record, except for the last record of the message.
	• If a position field is present and indicates an initial or intermediate segment, TCAM transfers the rest of the message to the MCP when the next PUT or WRITE is executed for this output data set. If no position field is present, TCAM assumes that the end of the 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 for the output data set signals the end of the message.

Signaling End of File and End of Message

TCAM provides the capability of signaling the application program that the contents of the message currently being processed by the application program constitute the end of a logical file of data; after processing the work unit or work units in this message, the application program may take an exit to a user-defined end-of-data subroutine. Such a subroutine might close the input data set, cause a different type of application-program activity to begin, issue a GET or READ macro referring to a different process queue, etc.

The user indicates that the contents of the current message represent the final portion of a logical file of data by issuing a SETEOF macro in the outheader subgroup of the application-program Message Handler. SETEOF can be coded to execute conditionally based on the presence in the message header of a specified character string. When SETEOF executes, a bit is set in the prefix of the message, indicating that this is the last message in the file. When a message with this bit on in the prefix is transferred to the application program by GET or READ macros, TCAM notes that this is the last message in the file and remembers this fact. Execution of the first GET or CHECK macro following transfer of the entire end-of-file message to the application-program work area gives control to the subroutine specified by EODAD.

Upon entry to the user-specified subroutine, the registers contain the same data as before execution of the GET or CHECK macro, except that register 15 contains the address of the exit subroutine.

If the user returns from this subroutine to the subroutine issuing GET or READ macros, these macros will execute in a normal fashion.

If no SETEOF macro is executed and a GET or READ macro referring to an empty process queue is issued, the application program enters a wait state until a message arrives at the process queue for the application program. (If READ was issued, the wait state begins only when the CHECK macro is executed.)

If the SETEOF macro executes and no EODAD exit is specified, when the next GET or CHECK macro following transfer of the entire end-of-file message to the application

Work-Unit Type:			Record			Message		
Work-Unit Format:			Fixed	Variable	Undefined	Fixed	Variable	Undefined
Input Side (GET/READ)	Work-Unit Size Determined By:	LRECL field of input DCB	X			X		
		BLKSIZE field of input DCB		X	X		X	X
		length field of READ macro			X			X
		delimiter specified via TPROCESS macro		X	X			
		end-of-message	X	X	X	X	X	X
	Work-Unit Size Stored In:	field in SAM prefix		X			X	
		LRECL field of input DCB			X			X
Output Side (PUT/WRITE)	Work-Unit Size Determined By:	LRECL field of output DCB	X		X	X		X
		length field of WRITE macro			X			X
		field in SAM prefix		X			X	
	Delimiter Specified via TPROCESS Macro Inserted After Each Record			X	X			

Figure 28. Effect of a Work-unit's Type and Format on the Way in which TCAM Determines its Size

program is executed, a completion code of X'00000004' is returned by TCAM in register 15, and control returns to the application program. User code may check for this return code and take appropriate action.

If record processing is specified (by the absence of U in the OPTCD= operand of the output DCB macro), the user may indicate that this is the last record in a message being sent from the application program to the MCP by coding X'F2' in the position field preceding the record in the work area (see *Optional Fields in the Work Area* in this chapter for a description of the position field). If no position field is defined, the program may signal TCAM 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. (If message processing is specified, and no position field is provided, TCAM assumes that the work unit being processed constitutes the entire message.)

Coding TCAM's Data Transfer Macros

TCAM provides facilities for obtaining messages from the MCP for processing, and for obtaining messages from the MCP for processing, and for returning response messages to the MCP. Although the messages are received from (or sent to) remote stations over communication lines, the programmer uses QSAM (GET and PUT) or BSAM (READ, WRITE, and CHECK) macros for obtaining and sending messages. A TCAM Message Control Program performs the device-dependent input/output operations for the application program. The user specifies whether he wishes to use the GET/PUT or READ/WRITE/CHECK support in the MACRF= operand of the input and output DCB macro.

Since TCAM's GET/PUT and READ/WRITE/CHECK support is similar to that provided by OS, the TCAM application programmer is expected to be thoroughly familiar with the OS sequential access method (BSAM or QSAM) whose counterpart he is coding in the TCAM application program. This requirement implies a knowledge of the applicable contents of *Data Management Services* and *Supervisor and Data Management Macro Instructions.*

GET Macro Instruction (QSAM only)

The GET Macro:

- Obtains work units from the MCP for processing,
- May be coded more than once in an application program.

The GET macro transfers a single work unit from the MCP to an application-program work area. The size of the work unit transferred depends upon whether record or message processing is specified by the OPTCD= operand of the input DCB macro (*Specifying Application-Program Work Units* in this chapter details the differences between record and message processing).

The GET macro instruction has the following format:

Name	*Operation*	*Operand*
symbol	GET	dcbname[,areaname]

symbol

Function: Specifies the name of the macro instruction.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

dcbname

Function: Specifies the symbolic address of the data control block associated with the process queue from which the application program is to obtain a work unit.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: The DD statement for this DCB names a process entry in the terminal table coded especially to receive messages from the application program. See *Overview of the MCP/Application-Program Interface* in this chapter.

If register notation is used, the register number (2 through 12) must be enclosed in parentheses, and the address of the data control block must previously have been loaded into the register.

areaname

Function: Specifies the symbolic address of the user-defined area into which the work unit is to be placed.
Default: None. If move mode is specified in the MACRF= operand of the input DCB macro, this operand is required. Otherwise, specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the specified register number must be enclosed in parentheses and the address of the work area must previously have been loaded into the register (1 through 12).

This operand may be omitted if locate mode is specified in the input DCB macro, in which case TCAM obtains a work area from the application program main storage by issuing a GETMAIN macro instruction when the input data set is opened. After the first GET, TCAM returns the address of the work area in register 1. TCAM uses this same work area until termination.

PUT Macro Instruction (QSAM Only)

The PUT macro:

- Returns work units to the MCP after processing,
- May be specified more than once in an application program.

The PUT macro causes the processed message or message segment to be transferred from the work area specified to the MCP, where it is processed by the incoming group of the MH for the application program, and then placed on the destination queue for a particular destination. This destination may be specified either in the message header and subsequently checked by a FORWARD macro in the incoming group handling messages from an application program, or as an operand of the FORWARD macro, or in a special destination field in the work area that may be filled in by user code before the PUT is issued (see *Defining Optional Fields in the Work Area* in this chapter for a description of the destination field).

If a PUT is issued and the message queues data set on reusable disk or in main storage that contains the destination queue for the destination of the message is congested with traffic, the PUT does not execute, a return code of X'00000010' is returned in register 15, and control passes to the next instruction; in this case, the user may test the return code and re-issue the PUT.

The PUT macro instruction has the following format:

Name	*Operation*	*Operand*
[symbol]	PUT	dcbname[,areaname]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

dcbname

Function: Specifies the symbolic address of the data control block for the output data set.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: The DD statement for this DCB names a process entry in the terminal table that is coded especially to receive messages from the application program. See *Overview of the MCP/Application-Program Interface* in this chapter.

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.

areaname

Function: Specifies the symbolic address of the user-defined work area from which the work unit is to be transferred.
Default: None. If move mode is specified in the MACRF= operand of the output DCB macro, this operand is required. Otherwise, specification optional.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the register number specified must be enclosed in parentheses, and the address of the work area must previously have been loaded into a register 0 or 2 through 12.

If locate mode is used, this operand should be omitted. In this case, the address of a work area into which the next work unit is to be placed is returned in register 1 for the first PUT macro referring to this DCB. For more information on locate mode, see *Dynamic Work Area Definition* in this chapter.

READ Macro Instruction (BSAM Only)

The READ macro instruction causes a work unit to be moved from the MCP into a designated area of main storage in the application program. It differs from the GET macro in that control may be returned before the work unit is retrieved when READ is used, whereas with GET control is not returned to the application program until the work unit is in the work area. The READ input operation may be tested for completion using a CHECK macro instruction; once CHECK is issued, control is not returned to the application program until the work unit is in the work area.

An application program containing more than one READ macro should be designed so that each data event control block (DECB) generated by a READ macro is associated with one and only one process queue from OPEN to CLOSE (i.e., the *decbname* and *dcbname* operands of the 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). The user may specify only one DECB per process queue. This technique allows the user to determine the status of any process queue by merely interrogating the current completion code in the DECB. See the completion codes in the next section. (The DECB is a system control block; for information on the layout of this control block see *The System Control Blocks* publication.)

NOTE: Since only one DECB may be specified per process entry, multiple READ macros directed to the same process queue are not permitted. However, the user may achieve the effect of issuing multiple READ macros directed to the same process queue by coding a list and an execute form of the READ macro. He would code one list form of READ and several associated execute form READ macros. The list READ and all the execute READ macros would specify the same DECB. The list and the execute forms of the READ macro are explained in the *Supervisor and Data Management Macro Instructions* publication.

Example:
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 READ 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
```

The READ macro instruction has the following format:

Name	*Operation*	*Operand*
[symbol]	READ	decbname,SF, dcbname,areaname, {length / 'S'}[MF={L / (E,listname)}]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

decbname

Function: Specifies the name to be assigned to the data event control block (DECB) created as part of the macro expansion.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.

SF

Function: None. Must be coded for proper macro expansion.
Default: None. This operand is required.
Format: SF

dcbname

Function: Specifies the symbolic address of the data control block associated with the process queue being accessed.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the address of the data control block must previously have been loaded into a register 1 through 12, and the register number must be enclosed in parentheses.

The DD statement for this DCB names a process entry in the terminal table coded especially to receive messages from the application program.

areaname

Function: Specifies the name of the work area into which the work unit is to be placed.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the address of the work area must previously have been loaded into a register 2 through 12, and the register number must be enclosed in parentheses.

{length / 'S'}

Function: Specifies the sum of the length of the work unit to be read plus the length of any optional fields in the work area.
Default: 'S'
Format: *length* is an unframed decimal integer.
Maximum: 32760 for length.
Notes: This operand is coded only for undefined-format work units; it is ignored for fixed-and variable-format work units.

Note that S is enclosed in single quotes. If 'S' is coded, and an undefined-format work unit is to be processed, the number of bytes to be read is taken from the LRECL= operand of the input DCB macro; for undefined-format work units, 'S' is the default.

MF= {L / (E,listname)}

Function: Specifies the list or execute form of the macro.
Notes: Described in the OS publication *Supervisor and Data Management Macro Instructions.*

WRITE Macro Instruction (BSAM Only)

The WRITE Macro:

- Returns work units to the MCP after processing,
- May be specified more than once in an application program.

The WRITE macro instruction causes the contents of a work area in the application program to be moved to the MCP in the same manner as PUT. Control may be returned before the block is moved. The output operation may be tested for completion using a CHECK macro instruction. (See the next section for a completion code table.)

The destination may be specified either in the message header and subsequently checked by a FORWARD macro in the incoming group handling messages from an application program, or by an operand of the FORWARD macro, or in a special destination field in the work area that may be filled in by user code before the WRITE is issued (see *Defining Optional Fields in the Work Area* in this chapter for a description of the destination field).

The WRITE macro instruction has the following format:

Name	*Operation*	*Operands*
[symbol]	WRITE	decbname,SF, dcbname,areaname, {length} {'S'}

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

decbname

Function: Specifies the name to be assigned to the DECB created as part of the macro expansion.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.

SF

Function: None. This operand must be coded for proper macro expansion.
Default: None. This operand is required.
Format: SF

dcbname

Function: Specifies the name of the data control block associated with the destination queue.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the specified register number must be enclosed in parentheses, and the address of the data control block must previously have been loaded into register 1 through 12.

areaname

Function: Specifies the address of the area from which the work unit will be moved to the MCP.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the specified register number must be enclosed in parentheses, and the address of the work area must previously have been loaded into register 2 through 12.

{length}
{'S'}

Function: Specifies the sum of the length of the work unit to be transferred to the MCP plus the length of any optional fields in the work area.
Default: 'S'
Format: 32760 for length.
Notes: This operand is meaningful only for undefined-format work units; it is ignored for fixed- or variable-format work units.

Note that S is enclosed in single quotes. If 'S' is specified and an undefined-format work unit is specified, the number of bytes to be written is taken from the LRECL= parameter of the output DCB macro.

BSAM/TCAM Completion Codes

After the user has issued a READ or WRITE, and the TCAM READ or WRITE routine has completed execution, a completion code is placed in the ECB in the DECB associated with the READ or WRITE. The codes are as follows:

Hex Code	*Meaning*
7F000000	Normal completion. (READ and WRITE)
70000000	SETEOF macro executed in MCP. The work area does not contain a work unit. (READ only)
5C000000	Congested destination message queues data set. (WRITE only)
58000000	Work unit sequence error. (WRITE only)
54000000	Invalid message destination. (WRITE only)
52000000	Work area overflow. (READ only)
50000000	READ issued in conjunction with a POINT macro to retrieve a message; message not found.
40000000	Data on read-ahead queue.
02000000	End of queue condition (not SETEOF and no data in TCAM MCP for DCB).
01000000	Read-ahead queue empty, but destination queue not empty.

The primary use of these codes is for communication between the READ or WRITE and CHECK routines (see the next section). If a user prefers to issue a WAIT macro rather than a CHECK macro, he is responsible for testing the completion code. A completion code of X'70000000' indicates an end-of-file condition and requires CHECK to take the user's EODAD exit. Code '5C000000' indicates that the WRITE was not effective because the message queues data set for the destination is congested with traffic and cannot accept the work unit at this time. The user may issue another WRITE in this case. Codes X'52000000', X'54000000', and X'58000000' indicate error conditions and require CHECK to take the user's SYNAD exit. Completion code X'58000000' indicates that the output DCB macro associated with the WRITE macro specifies OPTCD=C and that the work-unit position field specifies the wrong type of work unit – for example, the position field might say that this work unit is the first portion of a message, but the position field for the last work unit processed did not say that it was the last portion of a message. Codes X'02000000' and X'01000000' indicate that the process queue has no data on it; when data is placed on the queue, the code is automatically changed from X'02000000' to X'40000000'. Code X'40000000' indicates that after a READ macro was issued and the process queue was found to be empty, some data was placed on the process queue. Another READ or a CHECK macro should be issued to bring in this data. If SYNAD is not specified, a return code of X'00000008' is sent to the application program in register 15.

NOTE: Neither the wait nor the complete bit in the DECB's ECB is set to 1 by the two "empty-queue" completion codes (X'02000000' and X'01000000'). This allows the user to wait on ECBs posted in this manner without first having to set the wait bit in the ECB to 0.

User code may test the ECB before issuing a CHECK macro; if the ECB contains code X'02000000', the user routine might engage in some other program activity rather than issue the CHECK macro and enter a wait state.

CHECK Macro Instruction (BSAM Only)

The CHECK macro instruction causes the application program to be placed in the wait state, if necessary, until the associated input or output operation (READ or WRITE) is completed. The input or output operation is then tested for errors. If no error occurred, control returns to the instruction following the CHECK macro instruction. If an error occurred, the routine specified by the SYNAD= operand of the input or output DCB macro is given control. If no error routine is specified and an error occurred, a return code of X'08' is sent to the user in register 15 after the CHECK macro.

A CHECK may be issued after each READ and each WRITE in the same order as the READ or WRITE macro instructions are issued. If data is available at READ time, it is moved at CHECK time into the work area, and the event control block (ECB) in the data event control block (DECB) is posted complete with a return code of X'7F000000' (the ECB is contained within the first four bytes of the DECB, on a fullword boundary). If there is no data available and a user-controlled end of file has not been generated (by the SETEOF macro), the application program CHECK macro waits for data. When data is available, CHECK causes data movement; when this has been accomplished, the application program receives control after the CHECK macro.

A WAIT macro may be issued rather than a CHECK by specifying the DECB address in the ECB= operand of the WAIT macro; this provides a multiple-wait capability (see below). The WAIT macro is described in the *Supervisor and Data Management Macro Instructions* publication.

The CHECK macro has the following format:

Name	*Operation*	*Operand*
[symbol]	CHECK	decbname

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

decbname

Function: Specifies the name of the data event control block created by the associated READ or WRITE macro instruction.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols and must be the same as the *dcbname* for the associated READ macro.
Notes: If register notation is used, the specified register number must be enclosed in parentheses, and the address of the DECB must previously have been loaded into a register 1 through 12.

If the user detects an empty-queue completion code in the DECB and does not wish to wait implicitly (CHECK) or explicitly (WAIT) he may do some other processing. After this processing, the completion code will have been altered if a message has been placed on the associated read-ahead queue. The user must issue either CHECK or another READ to cause the pending READ to complete. This technique requires one DECB per process queue.

Multiple-Wait Capability

The user may achieve a multiple-wait capability by issuing more than one READ macro to more than one process queue and then issuing a WAIT macro. In the list specified by the ECBLIST= operand of the WAIT macro, the user would place the addresses of the DECBs associated with the READ macros issued as well as any other DECBs associated with the application program. A message satisfying a pending READ macro would cause a completion code of X'40000000' to be placed in the associated ECB. After the WAIT macro, one or more CHECK macros (or perhaps READ macros) should be coded so that the data will be moved into the user's work area. For information on the WAIT macro, see *Supervisor and Data Management Macro Instructions.*

Example:
In the following section of code, the user issues READ macros specifying DCBs associated with three process queues (DCBA, DCBB, DCBC), and then issues a WAIT macro specifying one event and having an ECBLIST= operand pointing to a list of addresses of the DECBs associated with the READ macros. When a pending READ macro is satisfied, a completion code of X'40000000' is placed in the ECB associated with the READ, and the address of the EBC is placed in register 1. If only one event is specified, the user may issue a CHECK macro specifying register 1; this macro moves the message satisfying the READ into its own work area (AREAA if READA was satisfied, AREAB if READB was satisfied, AREAC if READC was satisfied).

Name	*Operation*	*Operand*
MULWT	CSECT	
READA	READ	DECBA,SF,DCBA,AREAA
READB	READ	DECBB,SF,DCBB,AREAB
READC	READ	DECBC,SF,DCBC,AREAC
	WAIT	1,ECBLIST=DECBLIST
	LA	1,DECBA
	TM	DECBA,COMP
	BO	CHEKIT
	LA	1,DECBB
	TM	DECBB,COMP
	BO	CHEKIT
	LA	1,DECBC
CHEKIT	CHECK	(1)
	PROCESSING CODE	
DECBLIST	DS	0F
	DC	A (DECBA)
	DC	A (DECBB)
	DC	XL1'80'
	DC	AL3(DECBC)
AREAA	DS	100F
AREAB	DS	200F
AREAC	DS	300F
COMP	EQU	X'40'
	END	

Figure 29. Example of Multiple-wait Capability

The instruction immediately preceding the last address in the list causes the high-order bit of the last entry to be turned on; this is an OS requirement.

Application Program Error Exits

The input and output DCB macros for TCAM-compatible application programs permit the application programmer to specify an exit to be taken when certain errors occur during transfer of data between the MCP and the application program. This is the SYNAD exit, specified in the SYNAD= operand of the DCB macro.

The open or closed user subroutine whose address is specified in the SYNAD= operand receives control when certain errors occur. The user subroutine receives input identical to that provided by QSAM and BSAM for their SYNAD exit (as explained in *Supervisor and Data Management Macro Instructions).* This implies that SYNADAF or SYNADRLS macro instructions may be issued in the SYNAD routine. The next section details the register contents on entry to the SYNAD routine and the contents of the status indicator field for the SYNAD routine, while the following section contains information on using the SYNADAF macro.

The SYNAD routine specified by an input DCB macro is given control if 1) message-type processing has been specified (by coding U in the OPTCD= operand), 2) the message to be transferred by the current GET or READ macro is larger than that portion of the work area available to it, and 3) no position field is specified for this work area (OPTCD= does not specify C). In his SYNAD routine, the user must close and reopen this set before issuing another GET or READ; otherwise, TCAM will not continue to function properly. The routine is entered after a GET or CHECK macro is issued. If the error condition occurs and SYNAD is not specified, TCAM returns a completion code of X'00000008' in register 15 following GET or CHECK. In this case, user code should close the data set.

The SYNAD routine specified by an output DCB macro is given control when one of two logical output errors occur. These are:

1. The position field contains a value that is invalid (not X'40', X'F1', or X'F2', or X'F3') or that indicates that the current position of the message is out of sequence (e.g., 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).
2. The destination name in the destination field is not a valid entry in the terminal table.

For BSAM, this exit is entered only from the CHECK routine. If SYNAD= is not specified, condition (1) above results in a completion code of X'00000008' in register 15, while condition (2) results in a code of X'0000000C'.

Input to the SYNAD Routine

Input to SYNAD from TCAM/SAM access method modules is compatible with SAM. Register contents on entry to the SYNAD routine are as follows:

Register	Bits	Meaning
0	8-31	Address of the data event control block (DECB) for BSAM; address of status indicators for QSAM.
1	0	Bit is on for error caused by GET or READ.
	1	Bit is on for error caused by PUT or WRITE.
	4	Bit is on if user specified an invalid destination (PUT or WRITE).
	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 error analysis routine specified by the SYNAD= operand of the input DCB.

Word five (5) of the DECB (DECB+16) contains the status indicator address for BSAM support. Status indicators for the SYNAD routine are as follows:

Offset from status Indicator address		*Meaning*
Byte	Bit	
+2	0	Command reject (work units out of sequence).
+13	1	Incorrect length (workarea overflow).

All other fields in the SAM-compatible status indicator field are unused by TCAM. Main storage for this block is allocated at OPEN time if the SYNAD= keyword is coded in the DCB macro instruction or if provision is made by an alternate source.

SYNADAF

If the user issues a SYNADAF macro specifying BSAM or QSAM in his error analysis routine, he receives the following values in the specified registers:

Register 1 contains 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 (see table below).

Register 0 contains a return code of X'00' right adjusted.

See *Supervisor and Data Management Macro Instructions* publication for further information on the use of SYNADAF and SYNADRLS.

Format of TCAM/SAM SYNADAF Message Buffer

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

Network Control Facilities

TCAM provides facilities for dynamically controlling the telecommunications network through macro instructions issued in an application program. Three macros are provided to allow the contents of a control block to be examined: TCOPY, ICOPY, and QCOPY. Two macros are provided to allow modification of the contents of a control block: TCHNG and ICHNG. TCAM also provides the MRELEASE macro, which releases messages queued for an intercepted station, and the MCPCLOSE macro (discussed in *Activation and Deactivation of the MCP Interface* in this chapter), which initiates closedown of the Message Control Program. These macros are described in detail below. The facilities provided by these macros are also available using the operator commands of the operator control facility (see each macro description below).

NOTE: In order to execute, TCOPY, QCOPY, and TCHNG require at least one open input or output DCB for this application program.

Application-Program Network-Control Macros

A) Interrogation Capability

- TCOPY — Copy the contents of a designated terminal-table entry and its associated option fields into a work area.
- ICOPY — Copy the contents of a specified line's invitation list into a work area.
- QCOPY — Copy the contents of the queue control block (and its related priority QCBs) associated with a terminal (or process entry) into a work area.

B) Modification Capability (Password Protection Optional)

- INTRO — PASSWRD={chars / 0}
  Defines the password.
- TCHNG — Place contents of work area into a terminal-table entry and its associated option fields.
- ICHNG — Replace a specified invitation list with the contents of the work area, or activate or deactivate all entries in the specified list.
- MRELEASE — Activate a destination for receipt of messages from the CPU.
- MCPCLOSE — Initiate termination of the TCAM message control program.

In addition to these macros, TCAM provides the user with the capability of defining his application program as a secondary operator control station (by coding the SECTERM= operand of the TPROCESS macro) and of entering operator commands from it by means of PUT or WRITE macros. Responses to these commands are sent to the destination specified by the ALTDEST= operand of the TPROCESS macro creating the terminal-table process entry associated with the PUT or WRITE macro. For more information on the use of an application program as an operator control station, see *Entering Operator Commands from an Application Program* in the chapter *Using TCAM Service Facilities.*

Protection against unauthorized use of the ICHNG, TCHNG, MRELEASE, and MCPCLOSE macros is provided by the PASSWRD= operand of these macros. The password specified must be the same as the password specified by the PASSWRD= operand of the INTRO macro, otherwise, the application program macro is ignored.

The user might code a special application program designed solely to modify the teleprocessing system in the event of errors or other unusual conditions. For example, he might code ERRORMSG or REDIRECT macros in an inmessage subgroup handling messages coming in over a line group; these macros could test various bits in the message error record and when these bits were on, the macros could direct a special error message or the message being handled when the error occurred to the process queue for the application program. The application program could fetch error messages by GET or READ macros, analyze them, and issue operator commands (if it were designated a secondary operator control station by the TPROCESS macro) or network control macros to modify the system in a manner appropriate to the error detected.

The user is required to have at least one open TCAM DCB in the application program in which these network control macros are issued.

TCOPY Macro Instruction

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.

TCOPY moves the contents of a designated terminal-table entry to a work area, together with the contents of any option fields that are associated with the entry. The terminal-table entry may be any of the six entry types.

Various functions of TCOPY are also provided by the STSTATUS and OPTFIELD operator commands (see the *Operator Commands* section in *Using TCAM Service Facilities).* Execution of TCOPY alters the contents of registers 14 and 15.

The dummy section (DSECT) describing the single, line, and group terminal-table entries has the following format:

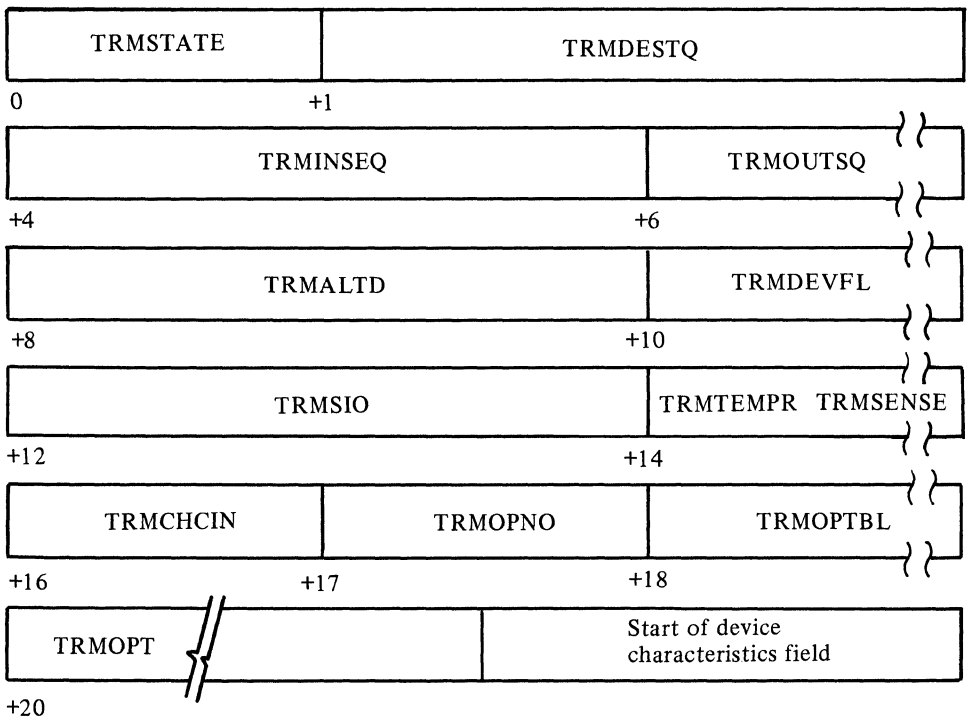


Figure 30. Terminal Table DSECT for Single, Line, and Group Entries

The length of the TRMOPT field 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 (if OPTION macros are coded) or the TRMCHCIN field (if no OPTION macros are coded). The first byte of each device-characteristics field contains the binary length of the rest of the field; the rest of the field contains the device-dependent data.

In addition to the contents of the terminal-table entry itself, TCOPY moves the contents of any option fields associated with a terminal-table entry into the specified work area. The first option field immediately follows the last device-characteristics field in the work area. A two-byte field named TRMOPTBL, located at an offset of 18 bytes from the beginning of the terminal-table entry, contains the offset from the beginning of the terminal-table entry to the beginning of the first option field in the user's work area.

The user must ensure that his work area is large enough to accommodate the largest possible string of data moved into it by TCOPY. (If the work area is not large enough to accommodate the data, the contents of main storage adjoining the work area are overlaid and lost.) The user may determine the length of the longest possible string of data that the TCOPY macro can move into his work area by looking at the assembly listing for his 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 CSECTs indicate the length of the terminal-table entry, the option-field offsets, and the device-characteristics fields, respectively. The user should find the sum of these lengths for each terminal-table entry he might wish to copy using TCOPY, and add to this sum the total length of the option fields associated with that entry. The work area named in TCOPY should contain a number of bytes equal to or greater than the largest sum obtained in this way.

One of the following return codes is returned to the application program in register 15 after the TCOPY macro is issued:

Code	*Meaning*
X'00000000'	The TCOPY macro executed successfully.
X'00000008'	TCAM is not in the system.
X'0000000C'	A TCAM application program DCB is not open; at least one input or output DCB must be open in order for this macro to execute.
X'00000020'	An invalid station name is specified in the field of the TCOPY macro (i.e., no such entry exists in the terminal table).

For a complete description of terminal-table entries, see the discussion of *Data Tables* in section 4, *Data Area Layouts* in the *TCAM PLM.*

The TCOPY macro has the following format:

Name	*Operation*	*Operands*
[symbol]	TCOPY	statname, areaname

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary).*

statname

Function: Specifies the name of the station whose contents are to be moved to the work area.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols and be the same as the name for the station specified in the MCP terminal table.
Notes: If register notation is used, the register must previously have been loaded with the address of a field containing the entry name. The name must be left-adjusted and padded with blanks to the length of the largest allowable station name. Permissible registers are 0, 2 through 12, 14 and 15.

areaname

Function: Specifies the name of the work area into which the contents of the terminal-table entry and its associated option fields are to be placed.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the register must previously have been loaded with the address of the work area. Framing parentheses must be coded. Permissible registers are 1 through 12, 14 and 15.

ICOPY Macro Instruction

The ICOPY macro:

- Permits examination of the contents of an invitation list.
- Is optional in a TCAM application program.

The ICOPY macro moves the contents of a designated invitation list to a work area. The function of ICOPY is not provided by the operator control facility. However, the ACTVATED and STATDISP operator commands cause display of the active and inactive terminals in a list and the status byte of an invitation list, respectively. Execution of ICOPY alters the contents of registers 14 and 15.

One of the following codes is returned to the application program in register 15 after the ICOPY macro is issued:

Code	*Meaning*
X'00000000'	The ICOPY macro executed successfully.
X'00000004'	An invalid relative line number is specified in the *rln* field of the ICOPY macro.
X'00000008'	TCAM is not in the system.
X'00000020'	The name specified is not the name of an opened TCAM line group DCB.

For a complete description of the invitation list, see *Defining Terminals and Line Control Areas* and in the *System Control Areas* discussion in Section 1: *TCAM PLM.*

Figure 31 below illustrates the format of an invitation list with three entries. Individual fields in the invitation list are discussed following the illustration.

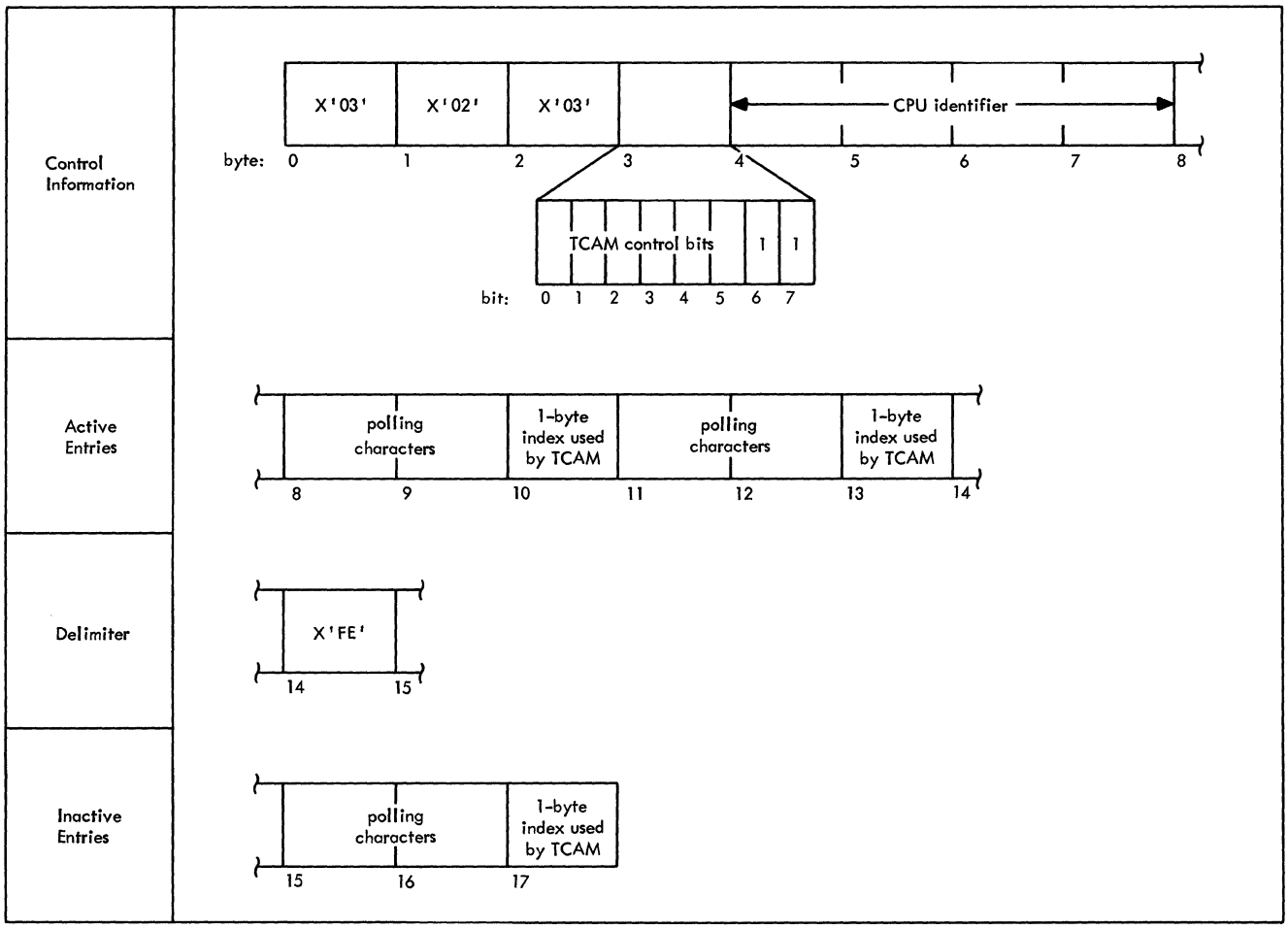


Figure 31. Sample Invitation List Containing Three Entries

Control Information:

Byte	*Meaning*
0	Indicates the total number of entries (both active and inactive) in this invitation list. All zeros in this byte indicates that this invitation list is for an output only line (stations on this line cannot enter messages). This invitation list contains three entries.
1	Indicates the number of active entries in this invitation list (an active entry is one that is currently eligible to be polled). This invitation list contains two active entries.
2	Indicates the number of bytes, including a one-byte index used by TCAM, in each entry in this list. The sample invitation list in the illustration above contains entries of three bytes each. The index byte must be the last byte in each entry.
3	Bits 0 through 5 in byte 3 are control bits used by TCAM (their contents must not be altered). If bit 6 is on, this is an active invitation list (it is being polled); if it is off, this invitation list is not currently eligible to be polled. If bit 7 is on, the Auto Poll feature is being used on the line corresponding to this invitation list; if it is off, programmed polling is in effect (this bit is meaningless if bit 6 if off). Bits 6 and 7 are both on in the sample invitation list, thus, this list is currently being polled by using the Auto Poll feature.
4-7	For non-buffered terminals, bytes 4 through 7 contain either all zeros or the address of a field that identifies the central processing unit into which TCAM is loaded. For buffered terminals, bytes 4 through 7 indicate the following:

Byte	*Meaning*
4	A one-byte count of the terminals on this line to which TCAM is currently sending.
5	Contains X'01' if the line is eligible for Auto Poll.
6	(unused)
7	A one-byte count of the total number of terminals on the line.

The contents of bytes 4 through 7 must never be altered.

Active Entries:

Byte	*Meaning*
8-10	Bytes 8 through 10 represent the first active entry in this invitation list. The polling characters for a station (or a component) are contained in the two-byte field starting at byte 8; although all the entries in this list use two-byte fields to contain polling characters, other lengths may be used. Byte 10 contains an index used by TCAM; this index must not be altered.
11-13	Bytes 11 through 13 represent the second active entry in this sample invitation list. (The same general discussion of bytes 8-10 also applies here.)

Delimiter:

Byte	*Meaning*
14	For an invitation list containing entries for BSC devices, an EOT character followed by X'FE' serves as a delimiter to indicate the end of the list of active entries. For start-stop devices, the delimiter is X'FE' without an EOT character. In this sample invitation list, entries are for start-stop devices. Two active three-byte entries precede the delimiter, and byte 0 indicates that there are three entries; consequently, there is a third entry in this invitation list (and since it follows the delimiter, it is an inactive entry). The contents of the delimiter must not be altered.

Inactive Entry:

Byte	*Meaning*
15-17	Bytes 15 through 17 represent the first inactive entry in this invitation list. (The same general discussion of bytes 8-10 also applies here, except that this is an inactive entry.)

The ICOPY macro has the following format:

Name	*Operation*	*Operands*
[symbol]	ICOPY	grpname, rln, areaname

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

grpname

Function: Specifies the name of the line group containing the line whose invitation list is to be displayed.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols and be the same as that specified in the DDNAME= operand of the DCB macro for the line group.
Notes: If register notation is used, the register specified must have previously been loaded with the address of a field containing the *grpname.* Permissible registers are 1-12, 14 and 15. Framing parentheses must be coded. The name must be left-adjusted and padded with blanks to eight characters.

rln

Function: Specifies the relative line number, within the line group, of the line whose invitation list is to be displayed.
Default: None. This operand is required.
Format: Unframed decimal integer greater than zero.
Maximum: 255
Notes: If register notation is used, the relative line number must previously have been loaded (in binary form and enclosed in parentheses) in the register designated. Permissible registers are 0, 2 through 12, 14 and 15.

areaname

Function: Specifies the name of the work area into which the designated invitation list is to be moved.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: The number of bytes to be allowed for each entry in the list depends upon the type of entry in the list.

If register notation is used, the register number specified must be enclosed in parentheses and must contain the address of the work area. Permissible registers are 2 through 12, 14 and 15.

QCOPY Macro Instruction

The QCOPY macro:

- Permits examination of a queue control block;
- Is optional in a TCAM application program.

QCOPY causes the contents of both a destination queue control block (QCB) and its related priority QCBs to be copied in a designated work area. The QCB is an internal TCAM control block associated with a destination queue. For a complete description of queue control blocks, see the discussion of *System Control Blocks* in section 4, *Data Area Layouts* in the *TCAM PLM.* A master QCB is 40-bytes long and always has associated with it at least one priority QCB (even if no priorities are specified for this destination QCB's corresponding station in the station's LEVEL= operand of its TERMINAL macro). Each priority QCB is 28 bytes long: therefore, the formula for determining the number of bytes needed in the work area in the user's application program is:

68 + 28n bytes

where *n* is the number of different priorities specified (for the station whose associated QCB is being copied) in the station's LEVEL= operand of its TERMINAL macro.

Part of the function of QCOPY is also provided by the QSTATUS and RLNSTATN operator commands (see their descriptions in the *Operator Commands* section of the chapter *Using TCAM Service Facilities).*

One of the following return codes is returned to the application program in register 15 after the QCOPY macro is issued:

Code	*Meaning*
X'00000000'	The QCOPY macro executed successfully;
X'00000004'	Invalid terminal-table entry type (e.g., distribution list, or cascade list is specified in the *termname* field).
X'00000008'	TCAM is not in the system.
X'0000000C'	A TCAM application program DCB is not open: in order for this macro to execute, at least one input or output DCB must be open.
X'00000020'	An invalid station name is specified in the *termname* field of the QCOPY macro (i.e., no such entry exists in the terminal table).

The QCOPY macro has the following format:

Name	*Operation*	*Operand*
[symbol]	QCOPY	termname, areaname

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary).*

termname

Function: Specifies the name of the terminal table entry whose QCB is to be displayed.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols and be identical to the name of the entry in the terminal table.
Notes: If register notation is used, the specified register number must be enclosed in parentheses and the register must contain the address of a field containing the name of the entry. The name must be left-adjusted and padded with blanks to the length of the largest allowable station name in the table.

areaname

Function: Specifies the name of the work area into which the contents of the designated QCB are to placed.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: If register notation is used, the register number designated must be enclosed in parentheses and must have been loaded previously with the address of the work area; permissible registers are 2 through 12.

TCHNG Macro Instruction

The TCHNG macro:

- Places specified data in a terminal table entry and its associated option fields,
- Is optional in a TCAM application program.

TCHNG causes the contents of a designated work area to replace the contents of a specified terminal-table entry. The TCOPY macro may be used to move the contents of a terminal-table entry to a work area where the contents are manipulated as desired (see the discussion of the TCOPY macro for a description of a terminal-table entry). The TCHNG macro is then used to move the modified entry back to the terminal table. Option fields are modified in the same manner by this macro.

All necessary information for proper execution of TCAM must be placed in the terminal-table entry in proper form. The contents of option fields may also be modified by the DATOPFLD operator command (see the *Operator Commands* section of this publication).

One of the following return codes is returned to the application program in register 15 after the TCHNG macro is executed:

Code	*Meaning*
X'00000000'	The TCHNG macro executed successfully.
X'00000008'	TCAM is not in the system.
X'0000000C'	A TCAM application program DCB is not open; at least one input or output DCB must be open in order for this macro to execute.
X'00000014'	Either a) an invalid protection password is specified as the PASSWRD= operand of the TCHNG macro, or b) the PASSWRD= operand is not specified in the TCHNG macro (and it must be specified because the INTRO macro's PASSWRD= operand specifies a protection password; code this operand exactly as it is coded in the INTRO macro).
X'00000020'	An invalid station name is specified in the *termname* field of the TCHNG macro (i.e., no such entry exists in the terminal table).

The TCHNG macro has the following format:

Name	*Operation*	*Operand*
[symbol]	TCHNG	termname, areaname [,PASSWRD=chars]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

termname

Function: Specifies the name of the terminal-table entry whose contents are to replaced by the contents of the designated work area.
Default: None. This operand is required.

Format: Must conform to the rules for assembler language symbols and be identical to the name of the station as specified in the terminal table.
Notes: If register notation is used, the specified register number must be enclosed in parentheses, and the register must contain the address of a field containing the name of the terminal-table entry, left-adjusted and padded with blanks. The field must be as long as the largest allowable station name with a maximum of eight characters. Permissible register are 2 through 12.

areaname

Function: Specifies the name of the work area from which the information is to be moved into the terminal table entry.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: The first byte of the entry receives the first byte of data in the work area, which must accordingly be the status byte. The work area should be at least as long as the longest terminal table entry that will be changed.

If register notation is used, the specified register number must be enclosed in parentheses, and the register must contain the address of the work area. Permissible registers are 2 through 12.

The new entry must contain in proper form all information necessary for successful operation of TCAM. See the description of the terminal table entries in the chapter *Defining Terminal and Line Control Areas.*

PASSWRD=chars

Function: Specifies the protection password that enables only qualified application programs to issue the macro.
Default: None. If the PASSWRD= operand of the INTRO macro was coded, this operand is required. Otherwise, specification optional.
Format: One to eight nonblank characters, unframed.
Notes: If coded, this operand must specify the same characters as were specified in the INTRO macro.

ICHNG Macro Instruction

The ICHNG macro:

- Modifies an invitation list;
- Is optional in TCAM application programs.

ICHNG causes the contents of a designated work area to replace the contents of a specified invitation list, or the stations in the specified list to be activated or deactivated for entering messages (if they are polled stations). ICOPY macro may be used to move the contents of an invitation list to a work area where the contents are manipulated as desired. The ICHNG macro may then be used to move the modified list contents back to the invitation list. For a complete description of the invitation list, see *Establishing Contact* in the chapter *Defining Terminal and Line Control Areas* in this book and the *System Control Areas* in Section 1 of the *TCAM PLM.* A sample invitation list containing three entries is presented in the discussion of ICOPY above.

If the macro is used to replace the contents of a specified invitation list with the contents of a work area, all necessary information for proper execution of TCAM must be placed in the invitation list in proper form. Entries in an invitation list may also be activated or deactivated by the ENTERING, NOENTRNG, NOTRAFIC and ACTVBOTH operator commands. The Auto Poll facility may be activated or deactivated by the AUTOSTRT and AUTOSTOP operator commands, respectively (if the autopoll bit is turned on in the UCB). See the description of these commands in the *Operator Commands* section of this publication. Stopping and starting of lines before and after changing the contents of an invitation list is handled automatically for the TCAM user. Execution of ICHNG alters the contents of registers 14 and 15.

One of the following return codes is returned to the application program in register 15 after the ICHNG macro is issued:

Code	*Meaning*
X'00000000'	The macro executed successfully.
X'00000001'	The DCB for the line group specified by *grpname* is not open.
X'00000004'	Either a) an invalid name is specified for the *grpname* operand of the ICHNG macro, or b) an invalid relative line number is specified in the *rln* field of the ICHNG macro (i.e., no such relative number exists for the group).
X'0000000C'	TCAM is not in the system.
X'00000014'	The PASSWRD= operand is not specified or is specified incorrectly in the ICHNG macro (and it *must* be specified because the INTRO macro's PASSWRD= operand specifies a protection password; code this operand exactly as it is coded in the INTRO macro).

The ICHNG macro has the following format:

Name	*Operation*	*Operand*
[symbol]	ICHNG	grpname,rln,{areaname / ACT / DEACT},[,PASSWRD=chars]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

grpname

Function: Specifies the name of the line group containing the line whose invitation list is to be modified.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols and be identical to the name specified on the DD statement associated with the line group.
Notes: If register notation is used, the specified register number must be enclosed in parentheses and the address of a field containing the *grpname* must previously have been loaded into the general register specified. The name must be left-adjusted in the field and padded with blanks to a length of eight bytes. Permissible registers are 1 through 12 and 14.

If the DCB for the line group has not been opened, ICHNG is not executed and a return code of X'01' is set in register 15.

rln

Function: Specifies the relative line number within the line group of the line whose invitation list is to be modified.
Default: None. This operand is required.
Format: Unframed decimal integer greater than zero.
Maximum: 255
Notes: If register notation is used, the register number specified must be enclosed in parentheses, and the register must previously have been loaded with the relative line number in binary format. Permissible registers are 1 through 12 and 14.

{areaname / ACT / DEACT}

Function: Specifies the type of modification or the modification itself.
Default: None. This operand is required.
Format: areaname, ACT or DEACT. *areaname* must conform to the rules for assembler language symbols.
Notes: *areaname* specifies the name of the area that contains the new invitation list. The first byte of the invitation list receives the first byte of the data in the work area, which accordingly must be the first byte of the invitation list control word.

ACT causes the activation of all entries in the specified invitation list. DEACT causes deactivation of all entries in the specified invitation list. No further polling will occur until the list is reactivated by an ICHNG macro specifying ACT, or an ENTERING operator command.

Register notation may be used for *areaname.* If register notation is used, the specified register number must be enclosed in parentheses, and the address of the work area must previously have been loaded into the register specified. Permissible registers are 1 through 12 and 14.

If *areaname* is specified, the new invitation list must contain, in proper format, all information necessary for successful operation of TCAM. See the description of the ICOPY macro for the format of the control word and of an invitation list.

PASSWRD=chars

Function: Specifies the protection password that enables only qualified application programs to issue the macro.
Default: None. If the PASSWRD= operand of INTRO was coded, this operand is required. Otherwise, specification optional.
Format: One to eight nonblank characters, unframed.
Notes: If coded, this operand must specify the same characters as specified by the PASSWRD= operand of INTRO. If the characters do not agree, or if INTRO specified PASSWRD= but this macro does not, ICHNG does not execute.

MRELEASE Macro Instruction

The MRELEASE Macro:

- Releases messages queued for a destination,
- Reactivates a destination made inactive by a HOLD macro or a SUSPXMIT operator command.

The MRELEASE macro releases messages queued for a station. This macro has the same effect as the RESMXMIT operator command.

One of the following return codes is returned to the application program in register 15 after the MRELEASE macro is issued:

Code	*Meaning*
X'00000000'	The MRELEASE macro executed successfully.
X'00000004'	The MRELEASE macro did not execute because either a) an invalid station is specified in the *statname* field of the macro, or b) the station is already receiving its queued messages.
X'0000000C'	TCAM is not in the system.
X'00000014'	The MRELEASE macro did not execute because either a) the protection password specified in the PASSWRD= operand does not match the protection password specified by the PASSWRD= operand of the INTRO macro, or b) a protection password is not specified in the PASSWRD= operand of the MRELEASE macro (and it must be specified because the INTRO macro's PASSWRD= operand specifies a protection password). Code the PASSWRD= operand exactly as it is coded in the INTRO Macro.

The MRELEASE macro has the following format:

Name	*Operation*	*Operand*
[symbol]	MRELEASE	statname [,PASSWRD=chars]

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

statname

Function: Specifies the name of the station that is now to receive its queued messages.
Default: None. This operand must be specified.
Format: Must conform to the rules for assembler language symbols and be the same as the name of the terminal entry.
Notes: If register notation is used, the address of a location containing the name of the station must be placed in the general register 2- 12 that is indicated in parentheses. The name must be left-adjusted and padded with blanks to the length of the longest station name.

PASSWRD=chars

Function: Specifies the protection password that enables only qualified application programs to issue the macro.
Default: None. Specification optional.
Format: One to eight nonblank unframed characters.
Notes: If the PASSWRD= operand is specified on the INTRO macro in the MCP, this operand must be specified and must be the same as the INTRO value. If they do not match or if this operand is omitted but a value is specified for INTRO, the MRELEASE macro does not execute.

TCAM's Message Retrieval Facility

During the operation of a telecommunications system, it may be necessary to retrieve a message that has already been placed on a destination queue located in a message queues data set on reusable or nonreusable disk. Messages cannot be retrieved from main-storage only queues, or if message traffic flows between stations whose queues are not the same type. TCAM uses a combination of POINT with GET or READ macro instructions to retrieve the desired message. After the message has been retrieved, user code may process it as appropriate and direct it to a desired destination. The message may be retrieved, whether or not it has already been sent to its destination, provided that the entire message has been queued on disk at the time that the POINT macro is executed.

If the application-program work area is too small to contain the entire message, the next GET or READ macro referring to the same DCB retrieves the rest of the message if C is specified in the OPTCD= operand of the appropriate input DCB macro. If C is not specified, the SYNAD= exit is taken. If the application program does not wish to retrieve the rest of the message, it may so specify by issuing a POINT macro whose address operand points to a block containing the station name followed by a X'40' (see the description of the *address* operand of the POINT macro below).

POINT Macro Instruction

The POINT Macro:

- Returns a station identification to the application program.

The POINT macro is used in conjunction with GET or READ to identify a station by passing in a register or a field the station identification and the sequence number of the message to be retrieved. Registers that may be altered during execution of the POINT routine are 0, 1, 14, and 15.

One of the following return codes is returned to the application program in register 15 after the POINT macro is issued:

Code	*Meaning*
X'00000000'	The POINT macro executed successfully.
X'00000004'	No message having the specified sequence number is queued in the specified destination queue.
X'00000008'	The destination name specified is not a valid entry in the terminal table.
X'0000000C'	1) The specified destination queue is not located in a data set residing on disk; 2) More than one type of disk queuing is specified.

The POINT macro has the following format:

Name	*Operation*	*Operand*
[symbol]	POINT	dcbname,address

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

dcbname

Function: Specifies the name of the data control block in the application program for the subsequent GET or READ associated with the POINT macro.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols and must agree with the name of the DCB macro for the associated data control block.
Notes: If register notation is used, the address of the data control block must previously have been loaded in register 1, or 2 through 12. The register must be coded within framing parentheses.

address

Function: Specifies the symbolic address of a field needed for the POINT macro.
Default: None. This operand is required.
Format: Must conform to the rules for assembler language symbols.
Notes: *address* is the symbolic address of a field containing a block with three contiguous fields:

1. An eight-byte field containing the station name, left-adjusted and padded with character blanks (X'40').
2. Either I (X'C9') or O (X'D6') indicating either an input or output sequence number, respectively. This field contains a blank (X'40') for retrieval termination.
3. A two-byte binary sequence number, right-adjusted with leading binary zeros.

If register notation is used, the address of this area must previously have been loaded in a register 1 through 12. Framing parentheses must be coded.

TCAM's Inquiry/Rapid Response Facility

TCAM can maintain a connection between a station and an application program for a period of time not less than the duration of the message and its response. This feature is called lock mode, and is supplemented by a hardware feature known as conversational mode. In this mode, a station is able to accept a text response to an inquiry message without having to be selected prior to receiving the response. Lock mode and the conversational feature are complementary functions. Both shorten the interval between an inquiry and its response.

Lock mode is used for inquiry applications. For fastest response, the station remains on the line until an application program returns the required information. While the station is in lock mode, no incoming messages are accepted from any other station on the line, and no outgoing messages other than the response message are sent to any station on the line (including the station in lock mode). Many stations on other lines may be simultaneously locked to the same application program.

There are two types of lock mode – message lock and extended lock. The desired function is specified by an operand of the LOCK message handler macro instruction.

If the station is in message lock mode, the connection is maintained while the entire message is sent to an application program and until the response message arrives. The first message from the application program to arrive at the destination queue for the locked station is assumed to be the response. The line is automatically freed when the response has been sent.

In extended lock mode, the same station is polled again after the response has been sent to it. If the response is positive, the next inquiry message is entered by the station. If the response is negative, the station is repolled until a positive response is received. Lock mode is maintained until an UNLOCK macro is issued.

Once a station is in extended lock mode, all messages entered by it are assumed to be inquiry messages directed to the application program to which the station is locked. Destinations specified in the headers of messages and checked by a FORWARD macro are overridden when the station is in extended lock mode. Therefore, once extended lock mode is in effect, the FORWARD macro must be executed after the UNLOCK macro to be effective.

Message lock is used if a single inquiry will provide all the information required. For instance, an inventory application might handle inquiries requesting the quantity of a certain part in stock. Extended lock mode is used if a series of inquiries must be made, each requiring a response. In a credit application, the inquiries might ask if a person has an account, await verification, and then request the credit balance. (Extended lock mode is not supported for stations specifying TCAM's buffered terminal support using the BFDELAY= operand of their TERMINAL macros.)

Either form of lock mode may be entered unconditionally or conditionally. Conditional execution occurs when a message header containing a control character or character string is processed by a LOCK macro specifying that character.

The UNLOCK macro is used to remove a station locked to an application program from extended lock mode. It may also be issued either unconditionally or conditionally depending upon a control field specified by both macro and message header.

If a station locked to an application enters a message, and a quick closedown is initiated or the line is stopped by operator control, the response is received before the station is deactivated. If the application program data set is closed, TCAM automatically disconnects from lock mode all stations locked to that application program using the deactivated data set.

The various forms of lock mode are summarized in the following table.

	MESSAGE FORM	*EXTENDED FORM*
CONDITIONAL	*Coded:* LOCK MESSAGE, A *Locked:* When LOCK is executed, if A is the next character in the header of the message currently being handled by inheader subgroup. *Unlocked:* When response has been sent.	*Coded:* LOCK EXTENDED, B *Locked:* When LOCK is executed, if B is the next character in the header of the message currently being handled by inheader subgroup. *Unlocked:* On execution of an UNLOCK macro.
UNCONDITIONAL EXECUTION	*Coded:* LOCK MESSAGE *Locked:* When LOCK is executed. *Unlocked:* When response has been sent.	*Coded:* LOCK EXTEND *Locked:* When LOCK is executed. *Unlocked:* On execution of an UNLOCK macro.

The conversational mode feature is specified by the CONV= operand of the **STARTMH** macro. When the computer receives a message from a station using this feature, instead of sending the normal positive acknowledgment, the computer sends a response message (from an application program) to the station. The station interprets this as a positive response. Transmission in this manner saves two line turnaround sequences.

If conversational mode is specified, a logical block of data being entered by a station is treated by TCAM as if it were a complete message. That is, an EOB or ETX line control character is assumed to be an EOT. Conversational mode may occur only for receiving and is operative only if the station is placed in lock mode when the message is processed by the inheader subgroup.

Conversational mode may be specified unconditionally (CONV=YES) or conditionally by the use of an option field. The CONV= operand specifies a bit setting and a one-byte option field. If any of the bits in the option field are on, conversational mode will be used for this message handler. (If the option field is longer than one byte, the first byte in the field is the one tested.)

An example of an MCP and an application program using message lock and conversational mode is shown in the *Sample MCPs* section.

There are several coding considerations for the three macro instructions involved in utilizing TCAM's inquiry/rapid response facility. They are based upon the type of station being used, the logic of the application program, and the interaction with other Message Handler macro instructions. These considerations are summarized below.

LOCK

- Suggested for audio terminals.
- Cannot be used in extended mode with any station using TCAM's buffered-terminal support.
- Should not be used if the logic in the application program requires that certain inquiry messages not be provided a response (when a station sends an inquiry in lock mode, an application program *must* send a response to the inquiry station, otherwise, the line may be lost either to the inquiring station, or to another station to which the application program may erroneously send the response).
- Requires the user to specify a destination in the destination field in the work area, and to code a FORWARD macro having a DEST= operand specifying DEST=PUT in the incoming group of the MH for the application program.
- If the originating terminal is an IBM 2260 Local terminal, the entire line group is placed in lock mode. If another 2260 Local in the line group attempts to enter a message, the read request is recognized and queued for later servicing. However, the message will not be received from the second 2260 until a response message has been sent to the originating 2260.
- If a station on a switched line breaks the line connection by hanging up while in extended lock mode, the line is unavailable for transmission to or from any stations. To render the line available for further transmission, issue a STOPLINE operator command for the line, and then issue a STARTLINE operator command to reactivate the line.
- If the CANCELMG macro is executed in the inmessage subgroup for a lock message, the lock is not broken, and the station will be repolled.
- If the HOLD macro is executed in the outmessage subgroup for a lock response, the lock is not broken, the terminal is not held, and the message will be retransmitted immediately (i.e., it will be sent twice). This can result in an infinite loop if the condition for HOLD is permanent and the line or station is inoperative.
- If a station is held by an operator command while in lock mode, or if a lock is initiated while the station is held, all lock responses will be sent as if the station were not held. No other messages will be sent until the station is released.
- No QTAM network control macro should be issued in an application program for a line on which is located a station locked to this application program by a LOCK macro. If this happens this line is lost to the system, and any line over which any operator command is entered after this condition occurs is also lost to the system.

UNLOCK

- When the UNLOCK macro is issued in the inheader subgroup handling inquiry messages being received from a station in extended lock mode, the message currently being handled is routed to the destination specified in its header, or by a FORWARD macro, if UNLOCK is issued before the FORWARD macro is issued. If UNLOCK is issued after FORWARD, the message is routed to the application program to which the originating station was locked.

STARTMH

- CONV=YES should be specified if IBM 1030 or IBM 1060 stations are included on lines handled by this message handler, since these lines do not have the capability of entering an EOT line-control character after their messages.
- CONV=YES should not be coded if any IBM 2780 station or IBM 2770 station using TCAM's buffered-terminal support, is included on a line handled by this Message Handler. If CONV=YES is coded in either of these cases, device hardware assumes an error after a block of data is entered, and retransmits the same block when next invited to enter data.

TCAM/SAM Compatibility

TCAM gives the user the capability of testing his application programs in a non-teleprocessing environment and then running them in conjunction with a TCAM MCP. (An example would be exercising the logic of a TCAM application program by using input from a card reader with output going to a printer). In many cases, the user can convert from a non-TP to a TCAM environment merely by changing the DD statements for his application-program data sets.

If you intend to run a TCAM application program in a non-TP environment, you should remember the following points:

1. The OPTCD= operand of the DCB macro has incompatible meanings in a non-TP and a TCAM environment. Therefore, this operand should be omitted from the DCB macro and specified if needed at execution time by the DCB= parameter of the DD statement. Test data for the non-TP environment should contain any optional fields that would be present in the work area if the program were run under TCAM, for the space in the work area allocated to optional fields to be filled.
2. The POINT macro must not be issued in a non-TCAM environment.
3. When issued in a non-TCAM environment, the TCOPY, ICOPY, QCOPY, TCHNG, ICHNG, MRELEASE, and MCPCLOSE macros merely place a return code in register 15 indicating that TCAM is not in the system, and pass control to the next instruction.
4. The DCB checkpoint exit is ignored in a non-TCAM environment.

Coordinating TCAM Checkpoints of the MCP with OS Checkpoints of the Application Programs

TCAM checkpoints of the Message Control Program may be coordinated with OS checkpoints of TCAM application programs by CKREQ macros issued in the application programs. The purpose of 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 is used to ensure coordination between application program and MCP, and also how a user-specified exit from the input or output DCB macro for the application program may be used for this purpose. For more information on the TCAM checkpoint restart facility, see the chapter *Using TCAM Service Facilities.* The OS checkpoint facility is described in *Advanced Checkpoint/Restart Planning Guide.*

When external files are updated by the contents of messages sent to an application program, coordination of the contents of the files, the application-program environment, and the messages being sent to the application program following a continuation restart might be achieved by using OS checkpoints and the CKREQ macro, as described below, in conjunction with "flip-flop" files set up to revert upon restart to their status as of the last OS checkpoint. Another possibility would be to specify CKPTSYN=NO in the TPROCESS macros for the application program and take an OS checkpoint each time that a file update occurred. If one file update per message were performed and one OS checkpoint per message were taken, upon restart the application program would have to check for one duplicate message in order to ensure that updating of the file would resume from the point of interruption.

NOTE: An OS checkpoint cannot be taken for an application program that is in an attached task.

In the following discussions, "system failure" is assumed to involve MCP failure. If the MCP fails, the application-program data sets are automatically closed; after the MCP is restarted, the user may restart his application program.

Failure of the application program need not be accompanied by failure of the MCP. In some applications, the user might wish to close down his MCP following abnormal termination of an application program, so that both might be restarted from the same point. See *Coordinating MCP and Application-Program Restarts* below for more on this topic.

Using the CKREQ Macro Instruction for Coordination

When a CKREQ macro is executed in an application program, a checkpoint request record is made in the checkpoint data set for each process queue to which a GET or READ macro can be directed by the application program. This record is used to update the MCP environment upon restart. The CKREQ macro causes sending to the application program after restart to begin with the last message marked serviced at the time the checkpoint request record was taken, rather than with the last message marked serviced before MCP closedown or failure.

The CKREQ macro is effective only for queues created by TPROCESS macros specifying CKPTSYN=YES. When a continuation restart is performed, normal scanning of the message queues (as described in the discussion of the TCAM checkpoint facility in the chapter *Using TCAM Service Facilities*) does not occur for message queues created by TPROCESS macros specifying CKPTSYN=YES. Instead, the message to be sent from the process queue to the application program following restart is determined by the contents of the last checkpoint request record made for that queue as the result of execution of a CKREQ macro. If CKPTSYN=NO is specified, the first unserviced message in the highest-priority group of messages on the queue is sent following restart.

When the CKREQ macro is used in an application program with low message traffic, the record resulting from it may be obsolete compared to the MCP environment (e.g., it may contain information pertaining to a zone that has been wrapped on a reusable disk). When this happens, messages are lost.

In order for the CKREQ macro to expand, a QSTART macro must be coded as the first macro of the application program. (The QSTART macro is ordinarily coded only for QTAM application programs that are to run under TCAM,but is also coded for ordinary TCAM application programs when the CKREQ macro is used).

The CKREQ macro has the following format:

Name	*Operation*	*Operand*
[symbol]	CKREQ	(no operands)

symbol

Function: Specifies the name of the macro.
Default: None. Specification optional.
Format: Must conform to the rules for assembler language symbols (see the *symbol* entry in the *Glossary*).

The CKREQ macro has no operands. Registers that may be altered during execution of the CKREQ macro are 0, 1, 14, and 15.

Upon completion, a return code is placed in register 15. Possible values are:

a) X'00000000' - checkpoint record (s) written on disk.
b) X'00000004' - no checkpoint record was written on disk for this request.

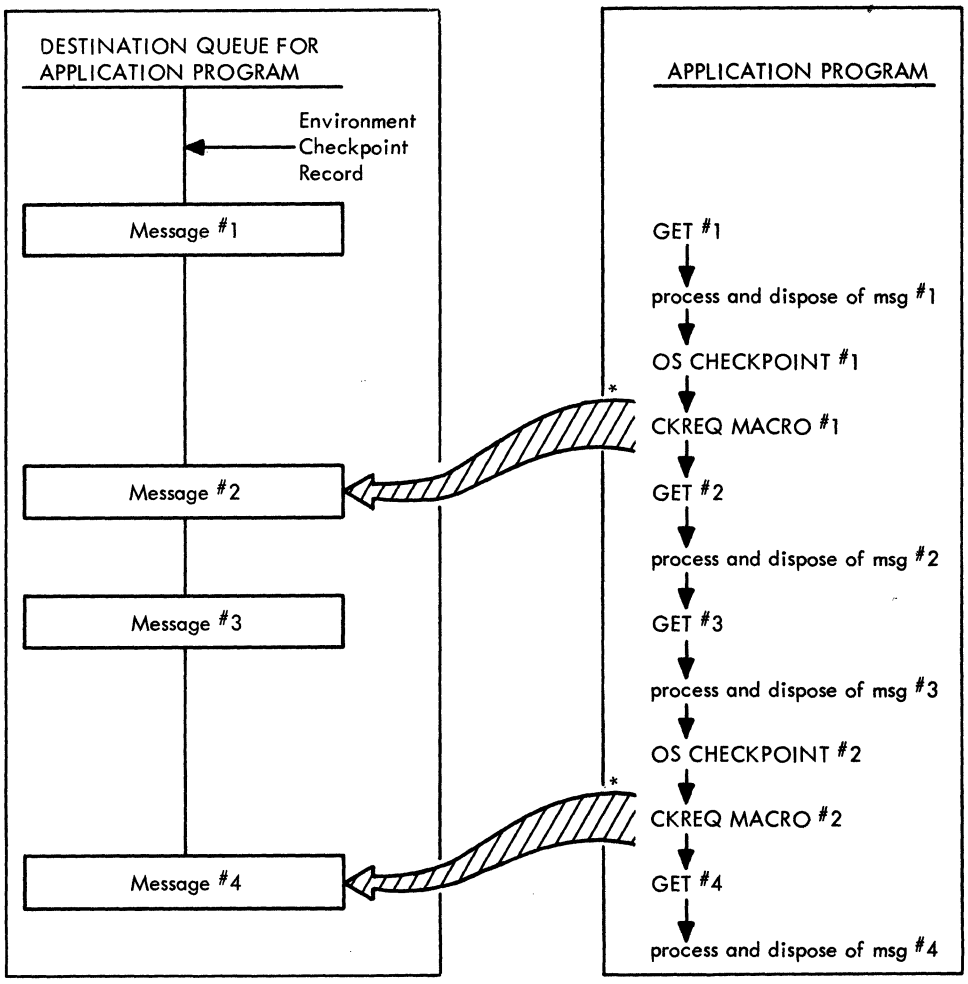
Suggestions for Using CKREQ

When the CKREQ macro is to be used to synchronize TCAM checkpoints of the MCP with OS checkpoints of the application program following system failure, CKPTSYN=YES should be coded for TPROCESS macros creating process queues to which the application program can direct a GET or a READ macro. A continuation restart should be specified in the STARTUP= operand of the INTRO macro.

After processing *n* messages or records, the user might take an OS checkpoint. After the OS checkpoint is taken, a CKREQ macro might be issued. If this were done, upon restart the application program environment would be restructured using the latest OS checkpoint and a maximum of *n* duplicate messages (i.e., messages already processed by the application program) would be sent.

NOTE: If both OS checkpoints and checkpoint request records are used, CKREQ should be issued each time an OS checkpoint is taken.

Figure 32 illustrates the use of CKREQ.



*indicates first message sent to application program from this queue following restart.

Figure 32. Example of Using the CKREQ Macro Instruction for Checkpoint Coordination

A TCAM environment checkpoint record is taken before GET No. 1 is issued. After the first message is processed and disposed of by the application program, an OS checkpoint is taken. Upon return from the checkpoint subroutine, a checkpoint-request record is

taken of the status of the destination queue for the application program. When GET No. 2 is satisfied, (i.e., after the second message has been moved into the work area) message No. 1 is marked serviced in the destination queue. When GET No. 3 is satisfied, message No. 2 is marked serviced. After message No. 3 is processed by the application program, another OS checkpoint record and TCAM checkpoint request record are taken. When GET No. 4 is satisfied, message No. 3 is marked serviced.

Assume that system failure (i.e., failure of the MCP) occurs during the processing of message No. 4. In this case, upon restart the application program would be reconstructed using OS checkpoint No. 2, and message No. 4 (the message pointed to by CKREQ macro No. 2) would be the first message sent upon restart. No duplication messages would be sent to the application program from this queue.

Now, assume that system failure occurs during processing of message No. 3. In this case, the application-program environment would be reconstructed using OS checkpoint No. 1, and message No. 2 would be the first message sent upon restart. This would be the next unprocessed message with respect to the reconstructed application program environment.

Finally, assume that system failure occurs after OS checkpoint No. 2 is taken, but before CKREQ macro No. 2 is executed. In this case, the application-program environment is reconstructed using OS checkpoint No. 2, but the first message sent upon restart is message No 2. Messages No. 2 and No. 3 would be duplicate messages with respect to the reconstructed application-program environment.

Using the DCB Exit for Coordination

The input and output DCB macros for TCAM application programs permit specification of a user-written routine to take an OS checkpoint after each TCAM environment checkpoint is taken. The user may specify the address of a problem-program exit list by coding the EXLST= operand of the input or output DCB macro for the application program. The list must start on a fullword boundary; its format and contents are discussed in *Data Management Services.* The user specifies his OS checkpoint routine by coding an X'0F' as a control-byte in the exit list and following the control byte with the three-byte address of his OS checkpoint routine. The user routine must save and restore the contents of registers 1 and 14. He must not store data in the area pointed to by register 13 upon entry to his routine. All registers except 1 and 14 contain what they held 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, the user should specify CKPTSYN=YES in the TPROCESS macro for each process queue to which a GET or READ may be directed.

When the EXLST= operand is coded, an indication is made to the application program each time an environment checkpoint record is made. If the EXLST= operand is coded in the input DCB macro, the first GET or READ macro issued by the application program after the environment checkpoint is taken passes control to the user-specified OS checkpoint routine. The GET or READ macro does not perform its function until after control is returned to the application program by the user routine. If the EXLST= operand is coded in the output DCB macro, the first PUT or WRITE macro issued by the application program after the environment checkpoint is taken passes control to the user-specified OS checkpoint routine. The PUT or WRITE is honored after control is returned to the application program by the user routine. If OS checkpointing is used, a CKREQ macro should be issued after every OS checkpoint.

Upon restart following system failure, message traffic to the application program resumes with the message in each process queue that was the earliest completed, unserviced message in the highest-priority group at the time the checkpoint was taken. Unserviced messages on the queue at the time the environment checkpoint was taken and all complete messages enqueued between the time the environment record was taken and the time of system failure are sent to the application program upon restart.

By coding the CPINTVL= operand of the INTRO macro, the user may ensure that environment checkpoints are taken within user-specified time limits.

NOTE: Ordinarily, the OS 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

Information on restarting the MCP after closedown or system failure is contained in *TCAM Checkpoint/Restart Facility* in the chapter *Using TCAM Service Facilities.* When restarting an MCP in conjunction with an application program, the MCP is restarted first. Then the application program is restarted using OS restart facilities.

If the MCP terminates abnormally, any TCAM application programs currently active are automatically terminated abnormally. If the TCAM checkpoint facility is being used, a continuation restart may be performed for the MCP; the application program may then be started.

If the application program terminates abnormally and the MCP does not terminate, the user has three courses of action open to him:

1. The user may restart his application-program job without closing down or terminating the MCP job. In this case, the first message received by the restarted application program from a particular process queue is that unserviced message in the highest-priority group for that queue which was completely received and enqueued before any other message in the highest-priority group in the queue. Remember that a message is not marked serviced on disk 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. 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 (assuming that the MCP continued to function between the time of application program failure and restart) would be message B. If this course of action is followed, no synchronization of OS checkpoints with the TCAM MCP is performed.
2. Following failure of the application program, the user can close down the TCAM MCP, then activate the MCP with a warm restart and the application program by an OS 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 taken for that queue, plus all messages that were placed on the queue after the last checkpoint request record was taken.
3. The user may cancel his MCP job. He would then activate the MCP by using continuation restart and the application program by an OS restart. In this case, the application program will receive all messages that were on the process queue and unserviced at the time the last checkpoint request record (or environment record, if no checkpoint request record was made) was taken, plus all messages that were placed on the queue after the last checkpoint request (or environment) record was taken.

When reusable disk queuing is used, there is an advantage to be gained from combining the two coordination methods described in this section by issuing both a CKREQ macro and an OS checkpoint request in the DCB exit routine (see the chapter *Defining Data Sets* for a discussion of reusable-disk queuing). If an environment checkpoint is taken due to a zone changeover on the reusable-disk data set, checkpoint request records taken prior to the data set reorganization are now out of date, because they do not point to the 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 take a fresh checkpoint request record after each zone changeover.

TCAM provides the user with a variety of facilities that support a teleprocessing system. Some of these facilities are specified by the user; others are provided automatically by TCAM.

Operator Control

The operator control facility enables the user to enter operator commands to examine or to alter the status of a telecommunications network. Operator commands may be entered from the system console, remote stations, and application programs; they are supported by either resident or nonresident routines (see *Appendix C*). A discussion of operator commands entered at the system console may be found also in the OS publication, *Operator's Guide.* The concepts and functions of operator commands for TCAM are discussed in *TCAM Concepts and Facilities.*

Initialization for Operator Control

Initialization for the operator control facility is accomplished through operands of the INTRO, TERMINAL, and TPROCESS macros. The INTRO macro specifies the single set of control characters to identify all operator commands (see the CONTROL= operand) and the primary operator control station (see the PRIMARY= operand); it also specifies the maximum number of command input blocks that may be used at any one time to contain operator commands entered at the system console (see the CIB= operand). The TERMINAL and TPROCESS macros associated with the stations selected as operator control stations have operands to indicate specification as secondary control stations (see the SECTERM= operand discussions of both macros). A *primary* operator control station receives the internally-generated error message, IEA000I, indicating that a permanent I/O error has occurred; it also has the capabilities of a secondary operator control station. (For a discussion of the IEA000I error message and permanent I/O errors, see *TCAM I/O Error-Recording Facility* in this chapter.) A *secondary* operator control station can send operator commands and can receive related responses, but not internally-generated error messages (with one exception: when a primary operator control station other than the system console becomes inoperative, message IEA000I is sent to the system console, in this instance a secondary operator control station, stating that the primary operator control station is inoperative).

General Format of Operator Commands

The fields of operator commands are separated by one or more blanks and must be in the order shown below. Commands entered at the system console do not include the *control chars* field and cannot occupy more than one line (e.g., if a command is entered through a card reader, it may not be more than 80 characters long, and if it is typed in at the system console, it may not be longer than 126 characters). Commands entered from either a remote station or an application program must not be longer than a buffer. Required letters (those shown in uppercase) must be entered in uppercase when an operator command is entered from either a station or an application program; if the command is entered at the system console, it may be either uppercase or lowercase. Brackets [ ] and braces { } are not coded. Brackets indicate an option; the enclosed item (or one of the several enclosed items) may be coded. Braces indicate that one of the several enclosed items must be coded.

```
control chars    operation operand    [nextline] ending
```

control chars

Used only with commands being entered either from a station or from an application program. Must be a character string (one to eight nonblank characters conforming to the rules for assembly language symbols) that identifies a command as an operator command (control characters are not recognized by the system if issued by the system console). One character string identifies all operator commands that may be entered from a station or an application program and is specified either at assembly time by the CONTROL= operand of the INTRO macro or at INTRO execution time by the L= keyword response to the WTOR message; the character string is stored in the AVT. User-written code in the MCP can override the CONTROL= operand to change this character string at the user's discretion (see the section on user code in the chapter *Designing the Message Handler).* The *control chars* field must be specified except when the operator command is entered at the system console (in which case *control chars*

must *not* be specified) and must be followed by one or more blanks. Command formats, in the *Operator Commands* section below, do *not* include the *control chars* field; however, each operator command that is entered from stations and application programs must begin with this character string.

operation

One of the following six operation *types* must be entered in the operation field by all sources requesting an operator command. One or more functions that provide system control are associated with each operation (see *Appendix F*):

- {VARY / V}
- {MODIFY / F}
- {HALT / Z}
- {DISPLAY / D}
- {HOLD / H}
- {RELEASE / A}

Braces indicate that a choice must be made in the form of the command (e.g., either VARY or V is keyed, not both; the shorter form is provided for coding efficiency).

operand

Entered by all sources requesting an operator command. This field consists of one or more operands (illustrated for each message in *Operator Commands*). These operands determine which functional operator command is associated with the operation *type* specified (*Appendix F*). If more than one operand is used, they are separated by commas without intervening blanks.

The most common operands used by the commands are *statname, address, grpname,* and *rln.*

statname

The name of the station, as specified in that station's TERMINAL macro.

address

The hardware address of the line, identical to the UNIT= operand of the DD statement for the line for which this operator command is being entered.

grpname

The name of the line group, identical to the DDNAME= operand of the DCB macro instruction for the line group for which the operator command is being entered.

rln

The relative line number of the line within the line group.

The first operand of all commands associated with the MODIFY operation has the following format:

{[procname.]id / jobname}

id is the abbreviation for *identifier.* The *[procname.]id* operand is used when TCAM has been started; it is identical to the *procname.identifier* field in the console START command. In an MVT environment, the *id* suboperand is a name chosen by the user in the START command for entering operator commands that are grouped as MODIFY operations, and *id* alone may be coded (the use of *procname.* is optional). In an MFT environment, the *id* suboperand is replaced by the partition number.

The *jobname* operand is used when TCAM is dequeued from the input stream (e.g., from a card reader). *jobname* is replaced by the name of the job to which the MODIFY operation applies, and is identical to the *jobname* field in the job statement for the job being modified by an operator command.

Thus, the TCAM job may be executed either as a normal job through SYSIN, or as a started procedure by the START command from the system console.

[nextline] ending

The simultaneous use of both subfields is restricted to *stations* entering operator commands (the *ending* subfield alone is entered when the command is from either the *system console* or an *application program*). *nextline* is replaced by the appropriate control characters needed to accomplish either a carriage return or a new line operation to ensure that TCAM's response message does not print over the previous line of print. See the IBM component description SRL (for the device that is used to enter the operator command) to determine the correct control character for either a carriage return or a new line operation. The *nextline* subfield, when used, is followed immediately, without intervening blanks, by the *ending* subfield. The *[nextline]ending* field must be separated from the operand field by one or more blanks; any invalid characters appearing between the blank delimiter and the first character of the *[nextline]ending* field are considered comments and are disregarded by TCAM. *ending* is the end-of-message signal and is required by all sources entering an operator command. The signal is EOB for the system console, EOT for start-stop stations, and ETX/EOT* for BSC stations. Command formats, in the *Operator Commands* section below, do not include the *[nextline]ending* fields; however, each operator command that is entered on the line must contain the appropriate code as described for it above.

*IF the CONV= operand of the STARTMH macro is specified (causing EOB, ETB, or ETX line-control characters to be treated like EOT line-control characters), then either EOB, ETX, or ETB alone is sufficient.

Specifying Operator Commands

The operator commands, together with the functions they perform, are discussed in a later section. Specification of these commands varies slightly depending upon whether the command is entered at the system console or from an application program or a remote station.

Commands entered at the system console follow the conventions outlined in the *Operator's Guide*. Required chraracters (those shown in uppercase) must be entered, but can be entered in either uppercase or lowercase, and the console operator must *not* enter the *control characters* field described above.

Commands entered either from an application program or a remote station must also enter required characters, and they must be uppercase unless the translation table used by the CODE macro, which recognizes the command, permits lowercase data. Furthermore, the *control characters* field must be the first field entered for each operator command for TCAM to recognize the operation.

Example 1:
If TCAM has been started and the *procname* field of the console START command specifies AQTPROC.QID and the INTRO macro specifies CONTROL=OPID, the command to change a terminal from secondary to primary operator control terminal status is:

- From an IBM 1050 terminal

OPID MODIFY QID,OPERATOR=NYC *nextline*EOT

or OPID F QID,OPERATOR=NYC *nextline*EOT

or OPID MODIFY AQTPROC.QID,OPERATOR=NYC *nextline*EOT

or OPID F AQTPROC.QID,OPERATOR=NYC *nextline*EOT

where uppercase characters are coded as shown (see the OPRIOPCL operator command in a later section, *Operator Commands*); NYC is the name of the station to be made primary, *nextline* is replaced by the appropriate *carriage-return* operation for the 1050, and EOT is the ending character to be used with the 1050.

- From an application program

  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

- From the system console

  MODIFY QID,OPERATOR=NYC EOB

  or F QID,OPERATOR=NYC EOB

  or MODIFY AQTPROC.QID,OPERATOR=NYC EOB

  or F AQTPROC.QID,OPERATOR=NCY EOB

Example 2:
If TCAM is being executed as a normal job through SYSIN with the jobname TCAMJOB and the INTRO macro specifies CONTROL=OPID, the commands of Example 1 become;

- From an IBM 1050 terminal

  OPID MODIFY TCAMJOB,OPERATOR=NYC *nextline*EOT

  or OPID F TCAMJOB,OPERATOR=NYC *nextline*EOT

- From an application program

  OPID MODIFY TCAMJOB,OPERATOR=NYC EOT

  or OPID F TCAMJOB,OPERATOR=NYC EOT

- From the system console

  MODIFY TCAMJOB,OPERATOR=NYC EOB

  or F TCAMJOB,OPERATOR=NYC EOB

Responses to operator commands are placed on the destination queue for the station that entered the command, and are sent through the outgoing Message Handler as normal messages. If selective execution is required in the outgoing groups, the first outheader subgroup can use a MSGTYPE macro to detect operator responses and a PATH macro to vary the processing path. Since responses to operator commands always begin with the character string IED, they are easily detected by macros such as MSGTYPE.

Operator responses are queued with no priority and without any line control information. For BSC lines and lines that expect internal blocking, the MSGFORM macro should be issued to provide the necessary blocking. The operator responses are placed on the destination queue in EBCDIC and must be translated to line code.

Assuming that the blocking operands and the translation table are specified in the DCB macro instruction for the line group of which the station accepting the operator response is a member, a suggested coding sequence is:

```
OUTHDR              FOR OUTGOING HEADERS
MSGTYPE C'IED'      DETECT OPERATOR RESPONSES
MSGFORM             ADD BLOCKING INFORMATION
CODE                TRANSLATE TO LINE CODE
PATH 1,PATHSW       SET A PATH SWITCH
MSGTYPE ,           HANDLE NON-OPERATOR RESPONSES
.
.
.
OUTEND
```

Possible responses to commands are included with each command description in the *Operator Commands* section. The responses described in *Incorrect Messages* (see below) may be returned in addition to one of those described for each command.

Entering Operator Commands from an Application Program

An application program may enter operator commands if SECTERM=YES is coded in the TPROCESS macro that creates the terminal-table process entry associated with the PUT or WRITE macro that moves messages.

When it wishes to enter a command, the application program moves that command into its PUT or WRITE work area (the operator command must have the same format as commands that are entered from stations; the command must begin with the *control characters* field).

A PUT or a WRITE macro is issued to move the command from the application program to the MCP. The inheader subgroup of the incoming group that handles messages entered by the application program must contain a CODE macro if the program may enter operator commands. One of the functions of CODE is to recognize operator commands (see the description of the CODE macro). Once it is recognized by CODE, the command from the application program is treated like any other operator command.

The response to a command entered from an application program is directed to the alternate destination specified by the ALTDEST= operand of the TPROCESS macro that creates the terminal-table process entry associated with the PUT or WRITE macro causing the operator command to be sent to the MCP. If no alternate destination is specified, the response is sent to the dead-letter queue. If no dead-letter queue is provided, the response is lost.

Incorrect Messages

The station operator may cancel a partially entered operator command by entering the *control characters* sequence (preceded and followed by one or more blanks) again after entering the initial control characters. At the system console, the CANCEL key is used. There is no response message for a canceled command.

Incorrectly formatted commands are returned to the operator, using a WTO response if the system console (or card reader) is being used. The format of the response message to an invalid command is:

IED016I STATION statname NOT FOUND

or

IED017I LINE { grpname,rln } NOT OPEN
             { address   }

or

IED018I operation COMMAND INVALID

where *statname, address, grpname,* and *rln* are explained in the operand description, and *operation* is the operation type, as specified in the operation field, and the first operand.

The response IED016I is received if the station name is not in the terminal table. This can be caused by a misspelled name, or by a name entered in lowercase when a folded translation table is not used.

IED017I is received if the line is not open, if the OPEN macro specified IDLE, if the groupname specified has no matching DD statement, or if the relative line number specified is zero or is higher than any relative line number in the group.

IED018I is received if the format of the command or a field in the command is incorrect. Possible errors include a required field missing or misspelled, fields in the wrong order, and numeric fields entered in non-numeric format.

Operator Commands

The operator commands appear in alphabetical order; command formats are discussed in an earlier section. See the discussions of *control chars* and *[nextline]ending* in an earlier section, *General Format of Operator Commands*, to determine the appropriate control characters that must precede operator commands that are entered from a station or an application program, and *carriage-return* (or *newline*) control characters and the end-of-message signal to be used with each command. Summaries of these command functions are illustrated in Figure 33 at the end of this section and in *Appendix F.* Keyword names (e.g., ACTVATED) are assigned to each command for ease of reference only (these names serve no programming function).

Where *statname* appears in an operand or a response message, it refers to the name of a station and must be identical to the name specified for that station's TERMINAL macro (see the definition of *symbol* in the discussion of the TERMINAL macro).

Likewise, *grpname* refers to a line group when it appears in an operand or a response message, and it must be identical to the name specified in the DDNAME= operand of the DCB macro for that line group.

Each response message to an operator command is preceded by a message number. The *Messages and Codes* publication contains a sequential list of all TCAM messages, including a complete discussion of each response to an operator command.

ACTVATED
This command requests a list of all entries in the invitation list for the specified lines that are currently active for entering messages (see also the descriptions of the INACTVTD operator command and the ICOPY macro instruction for an application program).

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,ACT,{grpname,rln / address}

Response:

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED036I {grpname,rln / address} ACTIVE={statname,... / NONE}

Explanation:
An operator command was entered to display the list of active stations associated with the line named by *grpname,rln* or *address* that entered. *statname,...* are the names of the entries that meet this requirement. If there are no active stations on the line, *statname,...* is replaced with NONE.

ACTVBOTH
This command combines the functions of the RESMXMIT and ENTERING operator commands and activates a nonswitched station for both accepting and entering messages. (Before issuing the ACTVBOTH command, a STOPLINE command must be entered to stop the line on which the station to be stopped or started resides. After ACTVBOTH is issued and its response is received, a STARTLINE command may be issued to restart the line.)

Format:

control characters	*operation*	*operand*
control chars	{VARY {V	statname,ONTP,B

statname is replaced by the name of the station to be activated. If a station is included more than once in an invitation list, all the entries for that station are activated.

Response:

IED016I STATION statname NOT FOUND

or

IED019I statname ALREADY STARTED

Explanation:
An operator command to start the station named by *statname* was entered. The station is already active.

or

IED020I statname STARTED

Explanation:
An operator command to start the station named *statname* was entered. The station is started, and the message is a confirmation of the action taken.

or

IED046I LINE FOR statname IS OUTPUT ONLY STATION

Explanation:
An operator command was entered to start a station for entering and accepting messages. *statname* is the name of the station to be started. The command is not processed.

or

IED089I LINE ACTIVE - VARY TERMINAL COMMAND REJECTED

Explanation:
An operator command to start a station was received, but the line for the station has not been previously stopped. The command is not processed.

AUTOSTOP
This command switches a line from the Auto Poll facility to the programmed polling facility if the automatic polling bit is on in the UCBTYP field of the UCB for the line (if this bit is on, the user gets Auto Poll at startup time).

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname}, AUTOPOLL={grpname,rln / address},OFF

Response:

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED027I AUTOPOLL STOPPED FOR {grpname,rln / address}

Explanation:
A request was made to stop autopolling on the line named by *grpname, rln* or *address.* This message confirms that autopolling has been stopped on this line.

or

IED028I AUTO POLL ALREADY STOPPED FOR {grpname,rln / address}

Explanation:
An operator command was entered to stop autopolling on the line named by *grpname,rln* or *address.* Autopolling on the line is not in progress at this time.

or

IED057I {grpname,rln / address} NOT CAPABLE OF AUTOPOLL

Explanation:
An operator command was entered to stop autopolling on the line named by *grpname,rln* or *address*, but according to the UCB for the line, the line is not capable of being autopolled or is a buffered station that is temporarily receiving. The command is not executed.

AUTOSTRT
This command changes a line from the programmed polling facility to the Auto Poll facility if the automatic polling bit is on in the UCBTYP field of the UCB for this line.

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname}, AUTOPOLL={grpname,rln / address},ON

Response:

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED021I AUTO POLL STARTED FOR {grpname,rln / address}

Explanation:
A request was made to start autopolling on the line named by *grpname,rln* or by *address.* This message confirms that autopolling has been started on this line.

or

IED022I AUTO POLL ALREADY STARTED FOR {grpname,rln / address}

Explanation:
A request was made to start autopolling on the line named by *grpname,rln* or by *address*, but autopolling is already active for the line.

or

IED057I {grpname,rln / address} NOT CAPABLE OF AUTOPOLL

Explanation:
An operator command was entered to start autopolling on the line named by *grpname,rln* or by *address*, but according to the UCB for the line, the line is not capable of being autopolled or is a buffered station that is temporarily receiving. The command is not executed.

CPRIOPCL
This command requests that either a secondary operator control station or the system console become the primary operator control station.

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname}, OPERATOR={statname / SYSCON}

statname is the name of a station other than the system console that is to become the primary operator control station. SYSCON must be coded if the system console is to become the primary operator control station.

Response:

IED016I STATION statname NOT FOUND

or

IED041I PRIMARY= {statname / SYSCON}

Explanation:
An operator command has been entered requesting that the station named by *statname* or the system console be made the primary operator control station. This message confirms that the requested action has been taken.

or

IED042I {SYSCON / statname} ALREADY PRIMARY

Explanation:
An operator command has been entered requesting that the station named by *statname* or the system console be made the primary operator control station, but the system status indicates that this is already so.

or

IED044I statname NOT ELIGIBLE FOR PRIMARY

Explanation:
An operator command was entered requesting that the station named by *statname* be made the primary operator control station, but *statname* is not eligible to be made primary (i.e., it is not defined as a secondary operator control station).

DATOPFLD
This command requests that data be inserted in an option field for a station.

Format:

control characters	*operation*	*operand*
control chars	{MODIFY} / {F}	{[procname.]id} , / {jobname} OPT=statname,opfldname,data

statname is the name of the station whose related option field is affected by this operator command. *opfldname* is the name of the option field, as specified in its related OPTION macro, where data is to be inserted. *data* is that data to be inserted into the specified option field. The data may either be enclosed in C' ', D' ', or X' ' framing characters or specified in unframed character format. The number of characters of data must be no greater than the size of the option field (if greater, it is rejected). If the replacement data is shorter in length that the size defined for the field, standard padding is used according to assembler language standards. All OPTION names are kept in a table with their offset into the offsets field of the terminal-table entry; this enables an option field named in an operator command to be found.

NOTE: Data to be inserted in an option field must be entered in the same format as the format defined for the option field. If the OPTION macro specifies a character data format, the data field of the macro may be either framed or unframed characters. If the OPTION macro specifies one of the hexadecimal formats (X, F, etc.) the data field of the command must be hexadecimal format with framing X' ' characters. If the formats do not agree, message IED056I is returned and the option field is not modified.

Response:

IED016I STATION statname NOT FOUND

or

IED034I statname HAS NO opfldname OPTION

Explanation:
An operator command was entered to modify the contents of the option field named by *opfldname* for the station named by *statname*, but no option field with this name exists for this station. The command is not executed.

or

IED050I statname OPTION opfldname MODIFIED

Explanation:
An operator command was entered to modify the contents of the option field named by *opfldname* associated with the station named by *statname*. This message confirms that the requested action is taken.

or

IED056I statname OPTION opfldname DATA FORMAT INVALID

Explanation:
An operator command was entered to modify the contents of the option field named by *opfldname* associated with the station named by *statname*, but the data format specified in the command differs from the definition of the option field format.

or

IED062I statname OPTION opfldname CANNOT ACCEPT SPECIFIED DATA

Explanation:
An operator command was entered to modify the contents of the option field named by *opfldname* associated with the station named by *statname*, but the data to replace the current setting of the option field is greater in length than the field.

or

IED077I statname OPTION opfldname DATA CHARACTER INVALID

Explanation:
An operator command was entered to modify the contents of the option field named by *opfldname* associated with the station named by *statname*, but the contents of the modification data do not agree with the framing characters surrounding the data.

DEBUG
This command activates a TCAM service aid routine that writes the dispatcher subtask trace table (STCB trace), the I/O interrupt trace table (line trace), or a dump of buffer and status information on either tape or disk. If buffers are written to the tape or disk data set using this command, it must be preceded by a DEBUG command that activates a line trace (and the line trace requires that the TRACE= operand of the INTRO macro instruction be coded). If the STCB trace is activated by this command, the DTRACE= operand of INTRO must specify a positive integer. Use of the DEBUG command requires that COMWRTE=YES be coded in the INTRO macro instruction.

NOTE: If either a closedown or a failure of the TCAM system occurs while DEBUG writing routines are still active, the functions provided by this command are not activated automatically when TCAM restarts. Reenter the DEBUG command with its appropriate operands after TCAM restarts to continue writing to the tape or disk data set. See *Debugging Aids* later in this chapter for more details on these and other diagnostic aids, including information on specifying the tape or disk data set and the separate utility that formats and prints the data set contents (the data set may contain various combinations of the STCB trace table, the I/O interrupt trace table, and buffers and status information).

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname},DEBUG={L / D}, {IEDQFE10 / IEDQFE20 / IEDQFE30}

Either L or D is coded in the DEBUG= keyword operand; L causes the service aid routine to be loaded and activated, and D causes the service aid to be deactivated and deleted. Either IEDQFE10, IEDQFE20, or IEDQFE30 is coded as shown. IEDQFE10 either activates (DEBUG=L) or deactivates (DEBUG=D) the service aid routine that writes the STCB trace table to either magnetic tape or disk, in order to use this service aid routine, the DTRACE= operand of the INTRO macro instruction must specify a

non-zero value. IEDQFE20 either activates or deactivates the routine that writes the I/O interrupt trace table; use of the DEBUG command for writing the I/O interrupt trace table requires that the TRACE= operand of the INTRO macro instruction specify a non-zero value, and the GOTRACE operator command must precede the DEBUG command (to activate the line trace). IEDQFE30 causes either activation or deactivation of the service aid routine that dumps TCAM buffers. See the chapter *Debugging Aids* for a description of the utility program to use to get a formatted listing of either the STCB trace, the line trace, or a buffer dump that is on tape or disk.

Response:

IED099I ROUTINE LOADED

Explanation:
The routine that was called by the DEBUG command is loaded and initialized.

or

IED100I ROUTINE DEACTIVATED

Explanation:
The routine designated in the DEBUG command was deactivated and deleted.

or

IED101I RESTART IN PROGRESS

Explanation:
The requested operation cannot be processed because TCAM is being restarted by either a checkpoint warm or a checkpoint cold restart.

or

IED102I INVALID OPERAND

Explanation:
The DEBUG command format is incorrect. One or more of the following operands were in error.

a. A subparameter other than L or D was specified.
b. An invalid routine name was specified. Valid names are:

IEDQFE10
IEDQFE20
IEDQFE30

or

IED103I ROUTINE ALREADY ACTIVE

Explanation:
A request has been made to activate a debugging routine that is already active.

or

IED104I ROUTINE NOT ACTIVE

Explanation:
A request has been made to deactivate a debugging routine that is not active.

or

IED107I COMWRITE NOT ACTIVE

Explanation:
A request has been made to activate a debugging routine that requires that the COMWRITE routine be active. COMWRITE is not active (COMWRTE=YES was not specified on the INTRO macro instruction).

or

IED124I QUEUE HAS BEEN WRAPPED

Explanation:
The message queues data set has been wrapped. Since the message queues data set can no longer be formatted reliably, the IEDQXB printing utility is terminating.

or

IED125I xxx BYTES NEEDED

Explanation:
Insufficient main storage exists for loading the requested debugging aid.

DPRIOPCL
This command requests the name of the current primary operator control station.

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,PRITERM

Response:

IED041I PRIMARY={statname / SYSCON}

Explanation:
An operator command was entered that requested the display of the station that is currently the primary operator control station. This response displays the requested information.

DSECOPCL
This command requests the names of current secondary operator control stations.

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,SECTERM

Response:

IED043I SECONDARY=statname,...

Explanation:
An operator command was entered that requested the display of all stations defined as secondary operator control stations. *statname,...* are the names of all stations so defined.

NOTE: All operator control stations except SYSCON are listed by this command, even if one of them is designated primary (a primary operator control station by definition has the capabilities of a secondary operator control station).

ENTERING
This command activates a terminal entry in an invitation list for entering messages from a nonswitched station that are to be received at the central computer. (Before entering this command, a STOPLINE command must be entered to stop the line on which the station to be stopped or started resides. After a response to the ENTERING command is received, a STARTLINE command may be entered to restart the line.)

Format:

control characters	*operation*	*operand*
control chars	{VARY / V}	statname,ONTP,E

statname is replaced by the name of the station to be activated. If a station is included more than once in an invitation list, all the entries for that station are activated for entering messages.

Response:

IED016I statname NOT FOUND

or

IED019I statname ALREADY STARTED

Explanation:
An operator command requesting the station named by *statname* be activated for entering has been received. The station is already active for entering.

or

IED020I statname STARTED

Explanation:
An operator command requesting that the station named by *statname* be activated for entering has been received. This response verifies that the requested action has been taken.

or

IED046I LINE FOR statname IS OUTPUT ONLY STATION

Explanation:
An operator command requesting that the station named by *statname* be activated for entering has been received, but the station is not capable of entering data. The requested action is not taken.

or

IED089I LINE ACTIVE - VARY TERMINAL COMMAND REJECTED

Explanation:
An operator command requesting that the station be activated for entering has been received, but the line associated with the station is active. The requested action cannot be taken.

ERRECORD
This command causes temporary-error records to be made for recoverable I/O errors occurring on a specified line or for a specified station. See the discussion of intensive-mode error recording in *TCAM I/O Error-Recording Facility* in this chapter for more information on the use of the ERRECORD command.

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname}, INTENSE=(LINE,{grpname,rln / address},sense,{count / 15} / TERM,statname)

grpname,rln, and *address*, respectively, are replaced by the name of the line group containing the line, the relative line number of the line within the line group, and the machine address of the line. *statname* is replaced by the name of the station for which failure incidents records are desired. Either LINE,... or TERM,... is coded, not both. (LINE,... provides intensive-mode error recording for all the stations on the specified line; TERM,... restricts intensive-mode error recording to the specified station.) *sense* is replaced by one of the following:

sense	*type of intensive recording provided*
BO	busout check
CR	command reject
DC	data check
EC	equipment check
IM	general intensive mode
IR	intervention required
LD	lost data
M2	leading graphics for 2740 Model 2 terminal
OR	overrun
TO	timeout
UE	unit exception

Eight of the conditions listed (busout check, command reject, data check, equipment check, intervention required, lost data, overrun, timeout) correspond to bits of the sense byte for the I/O device (which is, in this case, the transmission control unit being used). When the unit check bit is turned on in the CSW during an I/O operation, a sense command is issued by TCAM, and the appropriate bits in the sense byte are turned on. The CSW and the sense command are described in the *Principles of Operation.* A detailed discussion of the meaning of each bit in the sense byte may be found in the component description SRL manual for the transmission control unit being used.

Unit exception in the list refers to the unit exception bit of the CSW, which is turned on to indicate the presence of a condition that does not usually occur during an I/O operation.

When M2 is coded, a temporary-error record is made when an unusual leading graphic character (indicating a difficulty at the terminal) is received from an IBM 2740 Model 2 terminal, provided that the condition indicated by the character is recovered from. The use of leading-graphic *sense* characters by the 2740 Model 2 terminal, to indicate the terminal status and specific error conditions, is discussed in the publication *IBM 2740 Communication Terminal Models 1 and 2 Component Description* (Order no. GA24-3403).

When IM is coded, a temporary-error recording is made when any of the error conditions in this list (except for the unusual leading-graphics response for the IBM 2740 Model 2 terminal) occurs and is recovered from.

The *count* field is replaced by a decimal number from 1 to 15 (depending upon the number of records desired for the incident type specified in the *sense* field). If this field is omitted, a value of 15 is assumed.

Response:

IED016I STATION statname NOT FOUND

or

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED058I { grpname,rln / address / statname } SENSE COUNT=count, SETTING=sense

Explanation:
An operator command requesting that the sense information for the line named by *grpname,rln* or by *address* or for the station named by *statname* be altered is entered. This response verifies that the requested action has been taken.

GOTRACE
This command activates the TCAM I/O interrupt trace facility (line I/O trace) for a line. Use of this command requires that a positive integer be specified in the TRACE= operand of the INTRO macro instruction. See the *Debugging Aids* section of this chapter for information on the TCAM I/O interrupt trace facility.

Format:

control characters	*operation*	*operand*
control chars	{MOFIDY / F}	{[procname.]id / jobname}, TRACE={grpname,rln / address},ON

The trace is started for the line specified either by *grpname,rln* or by *address.*

Response:

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED023I TRACE STARTED FOR {grpname,rln / address}

Explanation:
An operator command requesting that the I/O trace be started for the line named by *grpname,rln* or by *address* has been entered. This response confirms that the requested action has been taken.

or

IED024I TRACE ALREADY STARTED FOR {grpname,rln / address}

Explanation:
An operator command requesting that the I/O trace be started for the line named by *grpname,rln* or by *address* has been entered, but I/O trace was already active on the line when the command was received.

or

IED055I I/O TRACE CANNOT BE ALTERED

Explanation:
An operator command requesting that the I/O trace be started has been entered, but the trace facility was not defined for this execution of the TCAM system.

INACTVTD
This command requests a list of the inactive entries in the invitation list for the specified line (see also the descriptions of the ACTVATED operator command and the ICOPY macro instruction for an application program).

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,INACT,{grpname,rln / address}

A list of inactive entries is displayed for the invitation list specified either by *grpname,rln* or by *address.*

Response:

IED017I LINE{grpname,rln / address} NOT OPEN

or

IED037I{grpname,rln / address} INACTIVE={statname,... / NONE}

Explanation:
An operator command has been entered requesting a display of the names of all stations associated with the line named by *grpname,rln* or by *address* that are currently inactive. The response provides a list of the names of all inactive stations. If no stations are inactive, NONE replaces *statname,...* in the response.

INTERVAL
This command activates the system interval whose value is specified by the INTVAL= operand of the INTRO macro instruction (if the INTVAL= operand is not coded, this operator command does not affect the system). For more information on the system interval, see *System Interval* in *Defining Terminal and Line Control Areas.*

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname},INTERVAL=SYSTEM

Response:

IED011I SYSTEM INTERVAL CANNOT BE ALTERED

Explanation:
Either a system interval of zero or no system interval at all was specified in the INTVAL= operand of the INTRO macro or in the response to a WTOR message at INTRO execution time, and an operator command was entered to modify the value of the interval.

or

IED045I SYS INTERVAL ALREADY ACTIVE

Explanation:
An operator command was entered requesting that the system interval be activated, but the system interval was already in the process of being activated.

or

IED093I SET SYSTEM INTERVAL COMMAND ACCEPTED

Explanation:
An operator command was entered requesting that the system interval be activated. This response verifies that the command has been received and is being acted upon.

INTRCEPT
This command requests display of all stations in the system that are intercepted (an *intercepted* station is one to which transmission has been suspended by a HOLD macro or a SUSPXMIT operator command).

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,INTER

Response:

IED039I NO STATIONS INTERCEPTED

Explanation:
An operator command has been entered requesting a display of all intercepted stations in the TCAM system. There are no stations currently intercepted.

or

IED040I INTERCEPTED STATIONS=statname,...

Explanation:
An operator command has been entered requesting a display of the names of all intercepted stations in the TCAM system. *statname,...* are the names of the stations that are currently intercepted.

LNSTATUS
This command requests display of the status field and the message error record for the specified line.

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,LINE {,grpname,rln / address}

The status field and the message error record are displayed for the line specified either by *grpname,rln* or by *address.*

Possible responses in the LNSTAT=... field of the response message are:

RESPONSE	*MEANING*
BS	bisync line
CM	line in control mode
CR	continue or reset operation
DL	switched (dial) line
IM	receiving initiate mode message
LF	line free
MS	msggen/startup message
NR	negative response to polling
OC	operator control is stopping line
RC	recall being performed
RV	line in receive mode
SD	line in send mode
TB	EOT from a buffered terminal
TR	I/O trace active

If no bits are set in the status field, the response is NO BITS ON.

Possible responses in the ERR=... field of the response message are given below. Each response is a mnemonic corresponding to a bit of the message error record. See *Appendix B* for an explanation of the error indicated by the mnemonic. If no bits are set in the message error record, the response is NO BITS ON.

ABR - abort-BSC station
CDC - connect/disconnect error
CHR - channel error
CUR - control unit error
CUT - cutoff error
FMT- format error
FWD - forward error
HDR - header incomplete
HDW - hardware error
INV - id from station invalid
ISB - insufficient buffers
LER - line error
LST - message lost (overlaid)
MAX - main storage maximum passed
MIN - main storage minimum passed
MNS - message not sent/received
NOP - station inoperative
NTS - TSO not in system
OLT - on-line test not in system
ORG - invalid origin
SEL - selection error
SQH - sequence high
SQL - sequence low
TER - terminal error
TXT - text transfer error
UNR - undefined error
UNX - unit exception
USE - user error

For a more complete discussion of these bits and their meanings, see the *TCAM PLM.*

Response:

IED017I LINE{grpname,rln / address} NOT OPEN

or

IED032I {grpname,rln / address} LNSTAT=status,... ERR=error,...

Explanation:
An operator command was entered requesting display of the status field and the message error record for the line named by *grpname,rln* or by *address.* This response displays the requested information.

NOENTRNG
This command prevents the control program receiving messages from a specific non-switched station by deactivating that station's entry in the invitation list. Any message currently being received is completed. (Before issuing the NOENTRNG command, a STOPLINE command must be issued to stop the line on which the station to be stopped or started resides. After NOENTRNG is issued and its response is received, a STARTLINE command may be issued to restart the line.)

Format:

control characters	*operation*	*operand*
control chars	{VARY / V}	statname,OFFTP,E

statname is replaced by the name of the nonswitched station to be stopped from entering messages. If a station is included more than once in an invitation list, all the entries for the station are deactivated.

Response:

IED016I STATION statname NOT FOUND

or

IED025I statname ALREADY STOPPED

Explanation:
An operator command has been entered requesting that the station named by *statname* be stopped from entering messages. The station is already stopped from entering.

or

IED026I statname STOPPED

Explanation:
An operator command has been entered requesting that the station named by *statname* be stopped from entering messages. This response confirms that the requested action has been taken.

or

IED046I LINE FOR statname IS OUTPUT ONLY STATION

Explanation:
An operator command has been entered requesting that the station named by *statname* be stopped from entering messages, but the line is defined as an output only line. The station is not capable of entering messages, and the requested action is not taken.

or

IED089I LINE ACTIVE – VARY TERMINAL COMMAND REJECTED

Explanation:
An operator command has been entered requesting that the station be stopped from entering messages, but the line associated with that station is still active. The requested action cannot be taken.

NOTRACE
This command deactivates the TCAM I/O trace facility for a line (line trace). The TCAM I/O trace facility is discussed in the *Debugging Aids* section of this chapter.

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname}, TRACE={grpname,rln / address},OFF

The trace is stopped for the line specified either by *grpname,rln* or by *address.*

Response:

IED017I LINE{grpname,rln / address} NOT OPEN

or

IED029I TRACE STOPPED FOR{grpname,rln / address}

Explanation:
An operator command has been entered requesting that I/O trace be stopped for the line indicated by *grpname,rln* or by *address.* This response verifies that the requested action has been taken.

or

IED030I TRACE ALREADY STOPPED FOR{grpname,rln / address}

Explanation:
An operator command has been requested that I/O trace be stopped for the line indicated by *grpname,rln* or by *address,* but I/O trace is not currently active for the line.

or

IED055I I/O TRACE CANNOT BE ALTERED

Explanation:
An operator command requesting that I/O trace be stopped has been entered, but the I/O trace facility is not defined for this execution of the TCAM system.

NOTRAFIC
This command combines the functions of the SUSPXMIT and NOENTRNG operator commands and may be used to stop transmission both to and from a station on a non-switched line. (Before issuing the NOTRAFIC operator command, a STOPLINE command must be issued to stop the line on which the station to be stopped or started resides. After NOTRAFIC is issued and its response is received, a STARTLINE command may be issued to restart the line.)

Format:

control characters	*operation*	*operand*
control chars	{VARY / V}	statname,OFFTP,B

statname is replaced by the name of the station to be stopped from both accepting and entering messages.

Response:

IED016I STATION statname NOT FOUND

or

IED025I statname ALREADY STOPPED

Explanation:
An operator command requesting that the station named by *statname* be deactivated was entered, but the station is already inactive.

or

IED026I statname STOPPED

Explanation:
An operator command requesting that the station named by *statname* be deactivated was entered. This response confirms that the requested action has been taken.

or

IED046I LINE FOR statname IS OUTPUT ONLY STATION

Explanation:
An operator command was entered requesting that the station named by *statname* be deactivated, but the station is not one that is capable of entering and accepting messages. It may accept messages, but the line with which the station is associated is defined as an output-only station. The requested action cannot be taken.

or

IED089I LINE ACTIVE – VARY TERMINAL COMMAND REJECTED

Explanation:
An operator command requesting that the station be deactivated is entered, but the line with which the station is associated is active. The requested action cannot be taken until the line is deactivated.

OPTFIELD
This command displays the field that is reserved in an option table by an OPTION macro instruction issued for a station.

Format:

control characters	*operation*	*operand*
control chars	{ DISPLAY } { D }	TP,OPTION,statname,opfldname

Response:

IED016I STATION statname NOT FOUND

or

IED034I statname HAS NO opfldname OPTION

Explanation:
An operator command has been entered requesting the display of the contents of the option field named by *opfldname* for the station named by *statname,* but no such option field is defined for that station.

or

IED035I statname OPTION opfldname=entry

Explanation:
An operator command has been entered requesting the display of the contents of the option field named by *opfldname* for the station named by *statname. entry* is the contents of the field, displayed in the format in which it was defined.

All OPTION names are kept in an option table with their offsets in the offsets field of the terminal entry; this enables an option field named in an operator command to be found. *statname* is replaced by the name of the station whose associated option field is to be displayed; *opfldname* is replaced by the name of the option field in the option table and is identical to the name field of the OPTION macro instruction that reserved space in the option table for this station.

POLLDLAY
This command, which is used only for stations on a nonswitched line, requests a change in the duration of the polling delay specified for the line group in the corresponding line group DCB macro instruction.

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{ [procname.] id / jobname }, INTERVAL=POLL,statname,data

statname is replaced by the name of any station on the line group and *data* by the decimal number of seconds (not to exceed 255) to be used for the polling delay. The length of the polling delay is changed for the entire line group, not just for the station named in the *statname* field above.

Response:

IED016I STATION statname NOT FOUND

or

IED048I POLLING DELAY FOR statname=data

Explanation:
An operator command has been entered requesting that the value of the polling delay for the station named by *statname* and its associated line be modified to the value specified by data. This response confirms that the requested action has been taken.

or

IED061I POLLING DELAY FOR statname CANNOT BE ALTERED

Explanation:
An operator command has been entered requesting that the value of the polling delay for the line associated with the station named by *statname* be modified, but the line is defined as a dial line and has no polling delay. The command cannot be executed.

QSTATUS
This command requests display of the fields of a queue control block containing the number of messages queued, the queue status, and the priority levels permitted for either a line or a station queue.

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,QUEUE,statname

If queuing is by station, *statname* is replaced by the name of any station for which the information is desired. If queuing is by line, *statname* may be the name of any station on the line for which the information is desired.

number specifies the number of messages in the queue. Possible returns for the *status* field in the response message are:

SNDBUF - sending to a buffered terminal
NONEON - no status bits on
TWELVE - call delay is greater than twelve hours
DELAY - in the delay queue
BUFFRD - queue for a buffered station
TSOSES - TSO session in progress
RDPRIO - read has priority

Possible returns for the queue *type* field in the response message are:

DR - reusable disk queue
DN - nonreusable disk queue
MO - main-storage-only queue
MR - main storage queue with reusable disk backup
MN - main storage queue with nonreusable disk backup
NO - no queuing used

PRIORITY=integer... specifies each priority level in the LEVEL= operand of the TERMINAL macro instruction issued for either the station or the line (see the discussion of the LEVEL= operand of the TERMINAL macro). A response of PRIORITY=000 indicates that no priorities were specified in the TERMINAL macro instruction.

Response:

IED016I STATION statname NOT FOUND

or

IED031I statname QUEUE SIZE=number,QUEUETYP=type,STATUS=status,...

PRIORITY=integer,...

Explanation:
An operator command has been entered requesting the display of queue information for the station named by *statname*. This response displays the requested information.

RESMXMIT
This command releases intercepted messages queued either for a specified station or for the line on which the specified station is located.

Format:

control characters	*operation*	*operand*
control chars	{RELEASE / A}	TP=statname

statname is replaced by the name of the station for which the released messages are queued (or the name of a station on a *line* for which the released messages are queued).

Response:

IED016I STATION statname NOT FOUND

or

IED053I statname ALREADY RELEASED

Explanation:
An operator command has been entered requesting the release from intercept status of the station named by *statname,* but the station is not currently intercepted.

IED054I statname RELEASED,SEQ-OUT=integer

Explanation:
An operator command has been entered requesting the release from intercept status of the station named by *statname*. This response confirms that the requested action has been taken, and provides the output sequence number of the first message to be released.

RLNSTATN
This command requests the relative line number on which a station resides.

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,ADDR,statname

statname is replaced by the name of the station whose relative line number is sought.

Response:

IED016I STATION statname NOT FOUND

or

IED038I statname IS ON LINE grpname rln address

Explanation:
An operator command was entered requesting information regarding the line with which the station is associated. *statname* is the name of the station about which the information is requested, *grpname* and *rln* provide the group name and the relative line number, and *address* is the hardware address of the line.

IED090I statname IS NOT A SINGLE ENTRY

Explanation:
An operator command was entered requesting information regarding the line that the station named by *statname* is on, but *statname* is not a single entry and has no line group, relative line number or machine address.

STARTLINE
This command causes transmission either to begin or to resume on a particular line (or all the lines) in a line group.

Format:

control characters	*operation*	*operand*
control chars	{VARY / V}	{(grpname,rln) / grpname / address},ONTP

grpname is the name of the line group and *rln* the relative line number of the line within the line group. The *rln* may also be replaced by ALL. The (*grpname,rln*) form of the operand starts transmission on the line indicated (the framing parentheses must be coded if this command is entered at the system console); the *grpname* form starts all the lines in the line group. *address* starts the line and is the machine address of the line (*address* consists of three hexadecimal digits). If polling is used, an invitation list that is active for entering messages is a prerequisite for message reception. STARTLINE initiates polling, enabling, or preparing of input lines. This command may thus be used to activate a line or line group that was opened idle.

Response:

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED019I {grpname,rln / address} ALREADY STARTED

Explanation:
An operator command has been entered to start the line named by *grpname,rln* or by *address* or the line group named by *grpname* (with the optional *rln* specified as ALL). The line or line group is already active.

or

IED020I {grpname,rln / address} STARTED

Explanation:
An operator command has been entered to start the line named by *grpname,rln* or by *address* or the line group named by *grpname* (with the optional *rln* specified as ALL). This response confirms that the requested action has been taken.

or

IED049I OLT CONTROLS LINE {grpname,rln / address} COMMAND REJECTED

Explanation:
The operator command to start the line named by *grpname,rln* or by *address* has been entered, but the line is currently controlled by the on-line test facility. The command cannot be executed.

or

IED092I BISYNC ERROR – LINE {grpname,rln / address} CANNOT BE STARTED

Explanation:
An operator command to start the line named by *grpname,rln* or by *address* has been entered, but it is a BSC line with an error preventing it being started.

STATDISP
This command displays whether the Auto Poll feature is being used for a specified line.

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,LIST,{grpname,rln / address}

The status of the invitation list is displayed for the line specified either by *grpname,rln* or by *address.*

Response:

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED059I {grpname,rln / address} LIST STATUS={AUTOPL / NO BITS ON}

Explanation:
An operator command requesting display of the status of the invitation list associated with the line named by *grpname,rln* or by *address* has been entered. This response displays the requested information. A response of NO BITS ON indicates an invitation list that is polled using programmed polling or a list associated with a dial line.

STOPLINE
This command stops transmission of messages on a line or a line group. The last operand determines whether transmission stops at the end of the current message (C) or immediately (I).

Format:

control characters	*operation*	*operand*
control chars	{VARY / V}	{(grpname,rln) / grpname / address},OFFTP,{C / I}

grpname is replaced by the name of the line group, and *rln* the relative line number of the line within the line group or by ALL (parentheses must be coded as indicated in the *grpname,rln* form of addressing if this command is entered at the system console). *address* is replaced by the machine address of the line. Either C or I is coded as shown (C stops transmission at the end of the current message, I stops transmission immediately). If either (*grpname,rln*) or *address* is coded, transmission stops on the specified line; if *grpname* alone is coded, transmission stops on the whole line group.

Response:

IED013I STOP REQUEST FOR SELF – VARY COMMAND REJECTED

Explanation:
An operator command to stop a line was entered, but the line specified is that which is associated with the station that entered the command. The command will not be executed.

or

IED017I LINE {grpname,rln / address} NOT OPEN

or

IED025I {grpname,rln / address} ALREADY STOPPED

Explanation:
An operator command was entered to stop the line named by *grpname,rln* or by *address* or the line group named by *grpname* or by *grpname*, ALL, but the line is not currently active.

or

IED026I {grpname,rln / address} STOPPED

Explanation:
An operator command to stop the line or line group named was entered. This response verifies that the requested action was taken.

or

IED049I OLT CONTROLS LINE {grpname,rln / address} COMMAND REJECTED

Explanation:
An operator command to stop the line named by *grpname,rln* or by *address* was entered, but the line is currently controlled by the on-line test facility. The command cannot be executed.

STSTATUS
This command displays the station status, the input sequence number of the next message to be received from the station, the output sequence number of the last message sent to the station, and the current intensive mode recording status.

Format:

control characters	*operation*	*operand*
control chars	{DISPLAY / D}	TP,TERM,statname

statname is replaced by the name of the station for which the status is desired.

Response:

IED017I STATION statname NOT FOUND

or

IED033I statname STATUS=status,... INTENSE={sense count / NO} IN-SEQ=integer

OUT-SEQ=integer

Explanation:
An operator command has been entered requesting display of information related to the station named by *statname.* This response displays the relevant information.

The status field in the station entry is examined, and possible conditions that may appear in the status,... field are:

INTCEPT	Station is intercepted.
SCNDARY	Station is a secondary operator control station.
SNGLTRM	Station entry is either a single or a group entry.
PROCESS	Station entry is a process entry.
DISLIST	Station entry is a distribution list.
CASLIST	Station entry is a cascade list.
LINEENT	Station entry is a line entry.
OPTFLDS	Station has option fields defined.

INTENSE=sense count indicates that a specific type (*sense*) and number (*count*) of intensive recording have been specified by the ERRECORD operator command for failure incidents, where *sense* and *count* are the same as that specified in ERRECORD's operand fields (see ERRECORD for a description of the intensive mode recordings that may be made and the restrictions on the number of recordings to be made). INTENSE=NO indicates that intensive mode recordings for failure incidents have not been requested by the ERRECORD command. *integer* in both IN-SEQ and OUT-SEQ refers to input and output sequence numbers, respectively.

SUSPXMIT
This command suspends transmission to a specified station.

Format:

control characters	*operation*	*operand*
control chars	{HOLD / H}	TP=statname

The form that the suspension takes depends upon the form of the first HOLD macro executed following this command in the Message Handler for the station. If an interval is specified for HOLD, this command causes a suspension of traffic for that period of time (the station is said to be *intercepted).* If the RELEASE operand is specified in the HOLD macro, suspension is maintained either until another operator command is issued to release messages queued for the station (see the RESMXMIT command) or until an MRELEASE macro (or a QTAM RELEASEM macro) is issued in the application program for the station. If no HOLD macro is specified in the MH, this command is rejected. *statname* is replaced by the name of the station to which transmission is to be suspended.

NOTE: An intercepted station may still enter messages -- only traffic to the station is suspended.

Response:

IED016I STATION statname NOT FOUND

or

IED051I statname SET FOR HOLD, SEQ-OUT=integer

Explanation:
An operator command has been entered requesting that the station named by *statname* be held. This response verifies that the requested action has been taken, and provides the output sequence number for the first message which is held.

or

IED052I statname ALREADY SET FOR HOLD

Explanation:
An operator command has been entered requesting that the station named by *statname* be held, but the station is already held.

or

IED060I statname CANNOT BE HELD

Explanation:
An operator command has been entered requesting that the station named by *statname* be held, but the station cannot be held because it is associated with a main-storage-only queue, it is on a line that is not open or has been opened idle, or there is no HOLD macro in the system.

SYSCLOSE
This command initiates either a quick or a flush closedown of the system (for a discussion of quick and flush closedowns, see the discussion of closedown in the *Deactivation* section of the chapter *Activation and Deactivation of the Message Control Program*).

Format:

control characters	*operation*	*operand*
control chars	{HALT / Z}	TP,{QUICK / FLUSH}

Either QUICK or FLUSH is coded as shown (the user must determine whether he wants a quick or flush closedown).

Response:

IED063I CLOSEDOWN IN PROGRESS – HALT COMMAND REJECTED

Explanation:
An operator command has been entered requesting that the TCAM system be closed down, but a closedown is already in progress.

SYSINTVL
This command changes the duration of the system interval previously specified in the INTVAL= operand of the INTRO macro (if the INTVAL= operand is not coded, this operator command does not affect the system).

Format:

control characters	*operation*	*operand*
control chars	{MODIFY / F}	{[procname.]id / jobname},INTERVAL=SYSTEM,data

For a discussion of the system interval, see *The System Interval* in the *Defining Terminal and Line Control Areas* chapter.

data is replaced by the decimal number of seconds to be used for the system interval (65535 is the maximum number that can be specified).

Response:

IED011I SYSTEM INTERVAL CANNOT BE ALTERED

Explanation:
Either a system interval of zero or no system interval at all was specified in the INTVAL= operand of the INTRO macro or in the WTOR response at INTRO execution time, and an operator command was entered to modify the value of the interval. The interval cannot be altered.

IED047I SYS INTERVAL IS data

Explanation:
An operator command was entered to change the system interval to the value specified by data. This command verifies that the action has been taken.

AREA AFFECTED	TYPE OF FUNCTION	KEYWORD NAME	SPECIFIC FUNCTION
System	Change	CPRIOPCL	Changes primary operator control to another station.
		ERRECORD	Records recoverable and nonrecoverable failure incidents.
		INTERVAL	Changes to system transmission interval.
		SYSINTVL	Changes the duration of the system interval.
	Closedown	SYSCLOSE	Initiates system closedown.
	Display	DEBUG	Activates a routine that dumps control blocks.
		DPRIOPCL	Displays the name of the primary operator control station.
		DSECOPCL	Displays the names of all secondary operator control stations.
		INTRCEPT	Displays all intercepted stations.
Line Group	Change	POLLDLAY	Changes polling delay for a line group.
	Start	STARTLINE	Starts transmission on a line or a line group.
	Stop	STOPLINE	Stops transmission on a line or a line group.
Line	Change	INTERVAL	Changes to system transmission interval.
	Display	ACTVATED	Displays the names of all active stations on a line.
		INACTVTD	Displays the names of all inactive stations on a line.
		LNSTATUS	Displays status of a communication line.
		QSTATUS	Displays status of a message queue.
		STATDISP	Displays the status byte of an invitation list.
	Start	AUTOSTRT	Starts Auto Poll on a line.
		GOTRACE	Starts TCAM trace facility on a line.
		STARTLINE	Starts transmission on a line or a line group.
	Stop	AUTOSTOP	Stops Auto Poll on a line.
		NOTRACE	Stops TCAM trace facility on a line.
		STOPLINE	Stops transmission on a line or a line group.
Station	Activate	ACTVBOTH	Activates a station for both accepting and entering messages.
		ENTERING	Activates a station for entering messages.
	Change	CPRIOPCL	Changes primary operator control to another station.
		DATOPFLD	Inserts data in an option field for a station.
		ERRECORD	Records recoverable and nonrecoverable failure incidents.
	Display	ACTVATED	Displays the names of all active stations on a line.
		DPRIOPCL	Displays the name of the primary operator control station.
		DSECOPCL	Displays the names of all secondary operator control stations.
		INACTVTD	Displays the names of all inactive stations on a line.
		INTRCEPT	Displays the names of all intercepted stations.
		OPTFIELD	Displays an option field for a station.
		QSTATUS	Displays the status of a message queue.
		RLNSTATN	Displays the relative line number of a station.
		STATDISP	Displays the status byte of an invitation list.
		STSTATUS	Displays the status of a station.
	Resume	RESMXMIT	Resumes transmission to a station.
	Stop	NOENTRNG	Stops a station entering after current message is completed.
		NOTRAFIC	Stops a station accepting and entering after current message is completed.
	Suspend	SUSPXMIT	Suspends transmission to a station (HOLD dependent).

Figure 33. Operator Commands Classified by Areas Affected

Checkpointing Operator Commands

If the checkpoint DCB has been opened, incident checkpoint records are written when the following operator commands have been successfully processed:

ACTVBOTH
AUTOSTOP
AUTOSTRT
CPRIOPCL
DATOPFLD
ENTERING
ERRECORD
GOTRACE
NOENTRNG
NOTRACE
NOTRAFIC
POLLDLAY
RESMXMIT
STARTLINE
STOPLINE
SUSPXMIT
SYSINTVL

Display commands and unsuccessful operations are not checkpointed. Commands affecting the invitation list are checkpointed only if the STARTUP= operand of the INTRO macro included I.

TCAM I/O Error-Recovery Procedures

The TCAM MCP includes a comprehensive set of error-recovery procedures for dealing with the various types of input/output errors that may occur in a telecommunications environment.

Whenever an input/output error interruption occurs, the error-recovery procedures examine the sense byte for the transmission control unit, and also the channel status word (CSW). (The CSW is described in the publication *Principles of Operation*, while the sense byte is described in the component description SRL for the transmission control unit being used.) If either the sense byte or the CSW indicates an error condition, TCAM takes action appropriate to the type of error.

An *irrecoverable* error is one that is incapable of being corrected by program action (e.g., overrun on a write command). For such an error, TCAM sets the appropriate bit or bits in the message error record, causes a special operator awareness message to be sent to the primary operator control station, causes a permanent error record to be written on disk by TCAM's I/O error recording facility, and may cause the connection between the computer and the station to be terminated.

The *message error record* is a five-byte storage area assigned to a message. The bits of the message error record indicate the presence (when on) or absence (when off) of specific error conditions, and may be checked by error-handling macros in the inmessage and outmessage subgroups of the Message Handler. These macros perform such functions as generating error messages, and causing all messages queued for a station to be held on the queue (because, perhaps, the station is inoperative). The message erorr record, which handles text errors as well as I/O errors, is described in *Appendix B.*

The section of this chapter titled *TCAM I/O Error Recording Facility* describes the permanent error record and the operator awareness message. Both of these yield information helpful to the user in diagnosing and correcting hardware difficulties that result in permanent I/O errors.

The connection with the station may be terminated in one of three ways following an irrecoverable error. For multipoint polled stations, the computer polls the next station in the invitation list. For a switched station, the computer attempts to send the next message queued for the station; if there are no more messages in the queue, the computer gives the station a chance to enter data, then hangs up. (If the switched station is not in text mode when the error occurs, the computer hangs up immediately; the computer

will redial if it initiated the call in the first place and if there are any messages on the queue.) For nonswitched contention stations, the computer merely resets itself to send or to receive the next message.

If the I/O error is not irrecoverable, TCAM's error recovery procedures may attempt to recover from it, usually by retransmitting the block of data in which the error occurred (this is called *retrying* the block). If the station is in text mode, the block probably will be retried only if at least one of four operands (START=, CONT=, CONV=, LOGICAL=) of the STARTMH macro is coded. If none of these operands were coded, no retries are likely to be performed, and the error is handled as an irrecoverable error. (If none of these operands is coded, retries will be performed for text errors if the error recovery procedures have access to the entire block in error.) If the station is not in text mode, the error recovery procedures will retry regardless of what is coded in STARTMH.

Two retries are performed for start-stop stations, while six are performed for BSC stations. If these retries fail to correct the error, it is treated as an irrecoverable error, and the actions described above for irrecoverable errors are taken.

The user can keep track of the number of *temporary* errors (i.e., errors that are recovered from as a result of retries) for a specific station by TCAM's I/O error-recording facility, described in the next section.

TCAM I/O Error-Recording Facility

TCAM provides an I/O error-recording facility that creates records on disk when certain terminal-related I/O errors occur. The TCAM error-recording facility, which is an extension of the OS outboard recorder (OBR) and statistical data recorder (SDR) error-recording programs and operates in conjunction with TCAM's I/O error recovery procedures, helps to reduce the time that the TCAM system is inoperative by providing information useful in diagnosing line and terminal problems.

For each station for which a TERMINAL macro is issued, TCAM maintains (in the terminal table entry) two counters. One of these is a two-byte counter that keeps track of the approximate number of Start I/O (SIO) commands issued for the station or line (SIO commands issued as a result of retrying during TCAM's I/O error-recovery procedures are not reflected in the total count). The other is a one-byte counter that contains the number of temporary errors (defined errors occurring during SIO operations for which retry was successful) that have occurred since the last error record was written on disk. If the station for which an SIO operation is being performed in known, the counters in the terminal table entry for that station are updated. The counters in a line entry in the terminal table are updated only if the station for which the SIO operation is being performed is not known; the counters are reset each time their contents are recorded on disk.

Four types of I/O error records may be written on disk: permanent, temporary, overflow, and end-of-day. These are discussed in order in the next section.

Kinds of TCAM I/O Error Records

A *permanent-error record* is written on disk for each permanent I/O error. A permanent I/O error is either an irrecoverable error (i.e., an undefined, unanticipated I/O error for which TCAM provides no error recovery procedure), or an I/O error for which TCAM provides an error-recovery procedure and has tried several times to correct the error only to fail each time. Each permanent-error record contains the following information:

- Date
- Time
- Program ID
- Station name
- Type of record (i.e., permanent, temporary, overflow, end-of-day)
- Contents of SIO counter for this line or station (count is approximate)
- Contents of temporary-error counter for this line or station
- First CCW
- Failing CCW
- Channel/unit address
- CSW
- Sense byte data

- Device type
- Unit status
- Channel status

Information on the CCW and CWS may be found in *Principles of Operation*. Record fields are discussed in the description of the IFCEREPO program in *Utilities*.

A *temporary-error record* is made on disk whenever an error occurs that is specified for a particular line or station in an ERRECORD operator command, provided that TCAM's error-recovery procedures are successful in recovering from the error. If TCAM's error-recovery procedures are unsuccessful, a permanent-error record is made and a special message is sent to the primary operator control station; the contents of this message are described below. This record contains the same information as the permanent-error record. More information on temporary error records and their use is contained in the section titled *Intensive-Mode Error Recording*.

A *counter overflow record* is made when either the SIO counter or the temporary-error counter in a particular terminal table entry is about to overflow. This record consists of:

- Date
- Time
- Program ID
- Type of record
- Station name
- Contents of SIO counter for this station (count is approximate)
- Contents of temporary-error counter for this station
- Channel/unit address
- Device type

Once the record is put on disk, the counters in the terminal table entry are reset.

When a line group data set is closed, an *end-of-day record* is made for each station and line in the line group for which there is a terminal table entry. Each record contains the same information as the counter overflow record.

The section below titled *Gaining Access to I/O Error Records* describes how to get formatted, printed records.

Intensive-Mode Error Recording

A station or line in *intensive mode* is one for which a temporary-error record is created each time that a specified error occurs and from which recovery is made. A station or line is put in intensive mode by means of an ERRECORD operator command. In issuing an ERRECORD command, the user may specify one of the particular types of error checked for by TCAM (time-out, lost data, overrun, data check, equipment check, bus out, intervention required, command reject, unit exception, or unusual leading graphic response from an IBM 2740 Model 2 terminal) and also may specify the number of times (1 to 15) that a temporary-error record is to be made when the specified error occurs and is recovered from for this line or station. Alternatively, the user may specify in the ERRECORD command that a temporary-error recording be made if *any* of the above errors (except for the unusual leading graphics response from the 2740 Model 2 station) occurs and is recovered from; in this case he would also specify the number of times a recording is to be made.

Intensive-mode error recording may be specified either for a line or for a station. If a station is specified in the ERRECORD command, temporary-error records are created when the error specified in the command occurs for that station and from which recovery is made. If a line is specified, a temporary-error record is made each time the specified error occurs and is recovered from for any station on the line (i.e., all stations on the line are placed in intensive mode).

If a station is placed in intensive mode for one type of error, and an ERRECORD command specifying a different type of error is then issued for the station, the type of error specified by the second operator command is the one that causes temporary-error records to be made after it is issued. An ERRECORD command for a line overrides those

issued previously for stations on the line; that is, if an ERRECORD command, which specifies that temporary-error records be taken for data checks occurring for a station named NYC, is followed by an ERRECORD command that specifies that temporary-error records be taken for time-outs occurring for any station on the line, after the second command is issued temporary-error records will no longer be taken for data checks occurring for the station NYC. If a third ERRECORD command, specifying that temporary-error records be taken for data checks occurring for NYC, is now issued, a temporary-error record will be made each time a data check occurs for NYC, and each time a time-out condition occurs for any other station on the line.

Operator Awareness Message

The following message is sent to the primary operator control station when an I/O error occurs for which TCAM provides error recovery procedures, if the error recovery procedures are unsuccessful in correcting the error. (This message is in addition to the permanent-error record that is created on disk when such an error occurs.)

IEA000I I/O ERR,aaa,bb,cccc,ddee,ffgghhhh

aaa
is the line address in hexadecimal format.

bb
is the command code in hexadecimal format as specified in the failing channel program.

cccc
is the status bytes of the channel status word (CSW) as specified in the input/output block (IOB) in hexadecimal format.

dd
is the first sense byte as specified in the input/output block (IOB) in hexadecimal format.

ee
always zero.

ff
is the TP Op code as specified in the failing CCW in the channel program for the last retry attempt (in hexadecimal format).

gg
Is the TP Op code of the failing CCW for the first occurrence of the error.

hhhh
For stations on switched lines, if the station is known, *hhhh* is replaced by the last four dial digits, if assigned; if the station is not assigned dial digits, *hhhh* is replaced by the station's addressing characters. For stations on nonswitched lines, *hhhh* is replaced by polling characters for receiving operations and by addressing characters for sending operations. If the station is on a switched line and is unidentified to TCAM at the time the error occurred, *hhhh* is replaced by the polling characters for stations on this line; if no polling characters are assigned, *hhhh* is replaced by zeros. *hhhh* appears in hexadecimal format.

Gaining Access to Error Records

Permanent-error, temporary-error, counter-overflow, and end-of-day TCAM I/O error records are located in the SYS1.LOGREC data set. The user can gain access to these records by using the IFCEREPO system utility program; information on using this program to write TCAM I/O error records may be found in the *Utilities* publication.

Network Reconfiguration

Direct control of network reconfiguration during execution is available for the station operator and the application programmer through operator commands and application program macros.

By Operator Commands

Functions provided by operator commands (with the corresponding command name in parentheses) are:

- Starting and stopping transmission on a line or a line group (STARTLINE; STOPLINE),
- Starting and stopping automatic polling on a line (AUTOSTART; AUTOSTOP),

- Starting and stopping the TCAM trace facility on a line (GOTRACE; NOTRACE),
- Activating and deactivating a station for accepting and/or entering messages on a line (ENTERING; ACTVBOTH; NOENTRNG; NOTRAFIC),
- Suspending and resuming transmission to a station (SUSPXMIT; RESMXMIT),
- Changing primary operator control to another station (CPRIOPCL),
- Activating system interval (INTERVAL),
- Changing the duration of the system interval (SYSINTVL),
- Changing the polling delay for a line group (POLLDLAY),
- Inserting data in an option field for a station (DATOPFLD), and
- Initiating system closedown (SYSCLOSE).

An example of TCAM's flexibility provided by network reconfiguration through operator commands is changing the status of a defective terminal before performing a manual device reconfiguration. See *Operator Control* in an earlier section for further information on the use of (and more detailed descriptions of) these and other operator commands.

By Application Program Macros

Two macros are provided for changing the contents of control blocks during execution of the application program; TCHNG and ICHNG are used to modify the contents of a terminal table entry and an invitation list, respectively. Two other macros, MRELEASE and MCPCLOSE, reactivate a destination and initiate system closedown, respectively.

TCHNG
This macro, in conjunction with TCOPY, moves the contents of a terminal table entry to a work area (where the contents optionally may be changed), and then moves the modified entry back to the terminal table. Related option fields may be modified the same way. Execution of this macro causes an incident checkpoint to be taken. It includes the station status, sequence numbers, and option fields. At restart time if the message queues are scanned, the sequence numbers will be overlaid if larger sequence numbers are encountered in the message on the disk queues.

ICHNG
This macro, in conjunction with ICOPY, moves the contents of a specified invitation list to a work area (where the contents of the invitation list optionally may be changed), and then moves the modified list contents back to the invitation list. When ICHNG executes, TCAM automatically stops message transmission on the line so that these changes can be made; when the invitation list contents have been replaced, TCAM restarts the line. Execution of the macro causes an incident checkpoint to be taken if I is specified in the STARTUP= operand of the INTRO macro.

MRELEASE
This macro, whose primary function is to release messages queued for a destination, also reactivates the destination that has been inactivated by a HOLD macro issued in the MCP. Execution of this macro causes an incident checkpoint to be taken.

MCPCLOSE
This macro, when executed in a user-written termination routine, initiates closedown of the telecommunications system. Execution of this macro causes an environment checkpoint to be taken.

For a more detailed discussion of network reconfiguration using these macros, see *Network Control Facilities.* See the introductory section of *Writing Application Programs* for an overview of the various approaches to incorporating application programs into a system (the manner in which a systems programmer designs his system directly affects network reconfiguration during the execution of his program).

TCAM Checkpoint/Restart Facility

The optional TCAM checkpoint/restart facility allows the TCAM system to be restarted with minimum loss of message data following closedown or system failure. TCAM achieves this goal by periodically recording, in a special data set on disk, information on the status of each station, destination queue, terminal table entry, and invitation list in the system; when start-up after system closedown or failure occurs, TCAM uses this information to restore the MCP environment to its condition before closedown or failure. Upon restart, the terminal table, line, option table, invitation lists, and internal control blocks associated with stations and lines are restored to the condition they were in when the last checkpoint record was taken; outgoing message traffic to each destination resumes with the highest-priority unsent message.

The TCAM checkpoint/restart facility permits restoration of the MCP environment upon restart. The OS/360 advanced checkpoint/restart facility (described in the *Advanced Checkpoint/Restart Planning Guide* publication) may be used to perform a similar service for TCAM application programs. In designing the TCAM facility, certain features were included to permit TCAM checkpoints of the MCP to be coordinated with OS checkpoints of TCAM application programs, so that upon restart the entire TCAM system (MCP plus application programs) would be restored as nearly as possible to its condition at the time of system closedown or failure. These features are discussed in the section *How to Coordinate TCAM Checkpoints of the MCP with OS Checkpoints of the Application Programs* of the chapter *Writing TCAM-Compatible Application Programs.*

The checklist below lists the macro instructions and operands that must be considered when checkpoint/restart is included in the system.

Macro	*Operand*	*Comments*
INTRO	CPINTVL={integer / 1800}	Specifies the maximum number of seconds between environment checkpoints. Specify any value between 30 and 65535, inclusive. If this operand is omitted, CPINTVL=1800 is assumed.
	CPRCDS={integer / 2}	Specifies the number of environment records to be retained in the checkpoint data set at any one time. Specify any value between 2 and 75, inclusive. If this operand is omitted, CPRCDS=2 is assumed.
	STARTUP={C / CY / W / WY} [I]	Specifies the type of restart to be performed following closedown of the MCP or system failure. C causes a cold restart after a normal quick or flush close, and a continuation restart (including scanning of message queues) after system failure. CY causes a cold restart after either a quick or a flush close, or after a system failure. W causes a warm restart after a normal quick or flush close, and a continuation restart (including scanning of message queues) after system failure. WY causes a warm restart after either a quick or a flush close, and a continuation restart (without scanning the message queues) after system failure. I causes the status of each invitation list to be included in the checkpoint record (indicates whether the list is active, and if active, whether it is being automatically polled). This value may be specified only when the invitation list is being checkpointed. If this operand is omitted, a value must be specified in response to the WTOR message issued by INTRO at execution time.
	CKREQS={integer / 0}	Specifies the maximum decimal number of destination queues in use at any time for application programs that include a CKREQ macro. *integer* is the number of checkpoint request records to be set up in a checkpoint data set. The maximum that may be specified is 255. If this operand is omitted, 0 is assumed.
	RESTART={integer / 0}	Specifies the environment record used to reconstruct the MCP environment as it existed at the time of closedown or failure. 0 causes the latest record to be used, 1 causes the next-to-the latest record to be used, etc. The maximum value that may be specified is 255; however, the value specified must be less than the number of environment records kept (see the CPRCDS= operand above). If this operand is omitted, RESTART=0 is assumed.
checkpoint DCB	DSORG=TQ	Identifies the data set organization as that for the message queues or checkpoint data set. This operand may not be omitted.

	MACRF=(G,P)	Specifies that access to the data set is to be gained with GET and PUT macro instructions. This operand may not be omitted.
	[DDNAME=ddname]	Is the name that appears in the DD statement associated with the data control block. If this operand is omitted, the value must be provided by the user's problem program any time before the data control block exit at open time.
	[OPTCD=C]	Specifies that the data set is for the checkpoint records. If this operand is omitted, the value must be provided either by a DD card, or by the user's problem program any time up to and including the data control block exit at open time.
	[EXLST=address]	Specifies the address of the problem program exit list. This list must be provided if user label, data control block, or user ABEND exits are required. The list must start on a fullword boundary. The user ABEND exit is discussed in the last section of the chapter *Defining the MCP Data Sets.*
OPEN	(dcbname,(INOUT))	An OPEN macro instruction must be provided in order to open the checkpoint data set. If this macro instruction is omitted, the checkpoint/restart facility is not activated.
CHECKPT	(none)	When coded in an incoming group, causes an incident record to be made of the status and the option fields assigned to the originating station or application program; the record is taken after the entire incoming group has executed. When coded in an outgoing group, causes an incident record to be made of the status and the option fields assigned to the destination station or application program; the record is taken after the entire outgoing group has executed.
CKREQ	(none)	Coordinates TCAM checkpoints of the MCP with OS checkpoints of TCAM application programs. When executed in an application program, causes a checkpoint request record to be made in the checkpoint data set for each process queue to which a GET or READ macro can be directed by the application program; these records are used to update the MCP environment upon restart. This macro causes a message to be sent to the application program (after restart) beginning with the message following the last message sent to the application program when the checkpoint request record was taken, rather than beginning with the last message marked serviced. Expansion of the CKREQ macro requires that a QSTART macro be coded as the first macro in an application program; CKREQ is effective only for queues created by TPROCESS macros specifying CKPTSYN=YES.
QSTART	(none)	Must be the first macro in an application program that includes a CKREQ macro.
TPROCESS	CKPTSYN=YES	If an OS checkpoint of the application program is used in synchronization with the TCAM checkpoint, CKPTSYN=YES must be specified in the TPROCESS macro that corresponds to the application program. CKPTSYN=YES specifies that the destination queue to which the application program directs its GET or READ macros is *not* to be purged of serviced messages at restart. If this operand is omitted, the queue is scanned normally and updated at restart.
input DCB	[EXLST=address]	This operand must be specified if a user-written routine is to be given control to initiate an OS checkpoint of the application program. *address* specifies the address of the problem program exit list; the entry in the list is a fullword consisting of a control byte (X'0F') followed by the three-byte address of a user-written routine that initiates an OS checkpoint.
output DCB	[EXLST=address]	(The discussion of the EXLST= operand for the input DCB also applies to the EXLST= operand of the output DCB.)

How the TCAM Checkpoint Facility Works

Checkpoint records, containing the information necessary to reconstruct the MCP environment upon restart, are kept in the checkpoint data set on a DASD. Directions for defining this data set are contained in the chapter *Defining the MCP Data Sets*, while a formula for determining the amount of space to allocate for this data set is given below (see the section *How to Get the TCAM Checkpoint Facility*). The four types of records that may reside in the checkpoint data set are a control record, two or more environment checkpoint records, a series of incident checkpoint records, and one or more checkpoint request records.

Types of Checkpoint Record The *control record* is used internally by TCAM during restart and requires no user coding considerations.

Environment checkpoint records are used to record the total MCP environment; each environment checkpoint record contains information on the status of each message queue, terminal, line and (optionally) invitation list at the time the record was taken, and also includes the contents of the option fields for each station.

If the checkpoint/restart facility is specified, environment checkpoint records are taken automatically at certain points during the execution of the MCP.

1. At the beginning of execution (when the READY macro is executed).
2. When the area alloted to incident checkpoint records has been filled with data (see the discussion of incident checkpoint records below).
3. If a message queues data set on reusable disk is present, when a zone changeover occurs (see *Reusable Disk Queues* in the chapter *Defining the MCP Data Sets*).
4. During a quick or flush closedown (discussed in the chapter *Activating and Deactivating the Message Control Program*).
5. After the time interval specified by the CPINTVL= operand of INTRO has expired.

These automatic checkpoints, along with the automatic incident checkpoints discussed below are sufficient to ensure satisfactory restart of the MCP itself after system closedown or failure. If the user is synchronizing his TCAM checkpoints with OS checkpoints of the application program, he may wish to ensure that a TCAM environment checkpoint be taken once a certain time interval has elapsed since the last environment checkpoint (see the discussion of the use of the DCB exit for coordination in *How to Coordinate TCAM Checkpoints of the MCP with OS Checkpoints of the Application Program* in the chapter *Writing TCAM-Compatible Application Programs*). This is done by specifying the time interval in the CPINTVL= operand of the INTRO macro instruction. TCAM keeps track of the amount of time that has elapsed since the last environment checkpoint; when the amount of lapsed time equals the time interval specified in INTRO, an environment checkpoint record is taken.

The user specifies the number of environment checkpoint records he desires to keep in his checkpoint data set at any one time by coding the desired number in the CPRCDS= operand of the INTRO macro instruction. If CPRCDS=3 is coded, the three most recent environment checkpoint records are kept in the checkpoint data set. When a new checkpoint record is taken, it overlays the oldest environment record in the data set. Ordinarily (i.e., unless the RESTART= operand of the INTRO macro instruction specifies some integer other than 0), TCAM uses the most recent environment record in the data set to restructure the MCP environment for a restart. If, however, the latest record cannot be used (due, perhaps to a disk Read or Write error), TCAM informs the user of this fact by means of a WTO message at the system console and automatically attempts to use the next most recent record. If that record is also unusable, and if there is another environment record in the data set, TCAM issues another WTO and attempts to use that record.

The more environment records there are in the data set, the greater is the likelihood that the environment can be recreated for restart. However, the recreated environment becomes increasingly inaccurate as earlier and earlier environment records are used; when environment records earlier than the latest are used, certain incident checkpoint records that TCAM needs to reconstruct the environment are likely to be overlaid and therefore inaccessible (see the discussion of the incident checkpoint below, and also the example at the end of this section). Another pitfall exists when a message queues data set on reusable disk is present; if TCAM's restart routine drops back to an environment

checkpoint record that was taken so long ago that the disk containing the data set has been wrapped since the time the record was taken, successful restart is unlikely, since pointers used by the TCAM restart routine to scan the message queues will have been destroyed in this case (scanning is discussed below).

Incident records are used to record single changes in terminal status, line status, system status, and option fields; these changes occur as a result of execution of MH macros, certain TCAM-related application-program macros (TCHNG and ICHNG), and operator commands. Each change in station status (from active to inactive or vice versa) is recorded by means of an incident record if STARTUP=I is specified on the INTRO macro instruction. At cold restart time, each change in a station's option fields caused by a TCHNG macro instruction or a DATOPFLD operator command is automatically recorded on an incident checkpoint record. One incident record is made of terminal status, of sequence numbers, and of the contents of the option fields assigned to the origin or destination station or application program each time a message is processed by an incoming or outgoing Message Handler group containing a CHECKPT macro instruction; this record reflects changes in the station's option fields caused by processing of the message by the MH.

NOTES: The user performing an initial startup or cold restart specifies (in the STARTUP= operand of the INTRO macro) whether or not he wants his invitation lists to be checkpointed. Whatever the user specifies, with respect to checkpointing of invitation lists, prevails until another cold restart is performed. If he specifies no checkpointing of invitation lists at initial start-up time but asks for such checkpointing in his WTOR response at INTRO execution time during a warm or continuation restart, he gets no invitation list checkpointing. If the input or output sequence number currently assigned to a station is less than the maximum it has had since the last cold restart (because the counter has been changed by a TCHNG macro, or because the counter has "wrapped" from 9999 to 1), TCAM uses an algorithm to determine which number is restored during a warm or continuation restart. TCAM takes the smaller number and subtracts it from the larger number. If the difference is less than or equal to 5000, the larger number is restored; otherwise, the smaller number is restored. If TCHNG changes the sequence number upward from that possessed by the last message, the same algorithm is applied to determine which number is restored. Changes in MCP status caused by operator commands (e.g., from programmed polling to automatic polling, from one polling interval to another, from an active to an intercepted station) are recorded by means of incident checkpoints and are reflected in the next environment record; that is, each operator command (except for INTERVAL and SYSCLOSE) that varies, modifies, or alters the status causes incident records. An exception is as follows: if I is not coded in the STARTUP= operand of the INTRO macro, changes from active to inactive status and vice versa are not reflected upon start-up; the original station status, as specified in the INVLIST macro, is reassigned in this case.

Incident records are used to update the information contained in environment records at restart time unless the STARTUP= operand of the INTRO macro instruction specifies WY. The TCAM restart routine takes the information contained in the latest usable environment record (unless an integer other than 0 is coded in the RESTART= operand of the INTRO macro instruction) and updates it with the contents of all incident records taken since the environment record was taken.

The number of incident records that may be taken depends upon the amount of space allocated on the disk for the checkpoint data set (allocation is discussed below). When the checkpoint data set is opened, space is automatically allocated for the control record, the number of environment checkpoint records specified by the CPRCDS= operand of the INTRO macro instruction, one incident record, and the number of checkpoint request records specified by the CKREQS= operand of INTRO. Any remaining space in the data set is used to set up additional incident checkpoint records. When all the space allocated for incident checkpoint records has been filled with records, another environment checkpoint record is taken automatically. (For further information on incident checkpoints and how they interact with environment records, see the example at the end of this section.)

Checkpoint request records are taken as a result of issuing CKREQ macro instructions issued in an application program. They record the status of the application program's message queues, option fields, and sequence-number fields, and are used in much the

same way as incident records to update the environment record during restart (except that the latest checkpoint request record for each application-program queue is used to update the environment record even when the checkpoint request record is older than the environment record.) The number of checkpoint request records set up by the TCAM checkpoint facility is specified by the CKREQ= operand of the INTRO macro instruction, and should be equal to the maximum number of process queues that are active at any time for application programs that include a CKREQ macro instruction. Each checkpoint request is associated with a particular process entry. Checkpoint request macro instructions help synchronize TCAM checkpoints with OS checkpoints of the TCAM application programs; their use is discussed in the section *How to Coordinate TCAM Checkpoints of the MCP with OS Checkpoints of the Application Program* in the chapter *Writing TCAM-Compatible Application Programs.*

Scanning the Message Queues: In addition to updating the latest usable environment checkpoint record (or the record specified in the RESTART= operand of the INTRO macro instruction) with any incident records taken since the environment record, the TCAM start-up routine may perform a scan of the message queues in the message queues data set.

A *scan* of the message queues involves searching the queues from the point at which the environment record being used for restart was taken to the point of system failure; already sent messages are passed over, so that after restart occurs, sending of messages to each destination station or application program represented by a message queue resumes with the highest-priority unsent message that was completely received before system failure. Scanning of the message queues occurs only when a restart following system failure is being performed, and then only if a Y is not coded in the STARTUP= operand of the INTRO macro instruction.

When a message on a disk message queue has been completely transmitted to a destination station, or completely transferred by means of either GET or READ macro instructions to an application-program work area, a TCAM routine sets a special bit that marks the message on disk as serviced. For a message transmitted to a station, the service bit is set when acknowledgment is received from the station that the entire message has been successfully received. For a message sent to an application program, the service bit is set when TCAM satisfies either a GET or a READ macro instruction for the next message in the queue; i.e., a message is not marked serviced until the next message has been entirely moved into the application-program work area. In performing a scan, the TCAM restart routine starts with the earliest message placed on the queue (or the earliest message that has not been overlaid, if the queue is on reusable disk) and goes down the queue to the point of failure; each message with a service bit on is passed over, while each complete message with a service bit off is transmitted according to its message priority. Messages on the same destination queue and having the same message priority are sent on a first-in first-out (FIFO) basis; that is, the message whose first segment arrived at the queue first is sent first, the message whose first segment arrived at the queue second is sent second, etc. (see also the discussion of message priority in the chapter *Defining Terminal and Line Control Areas*).

When a scan is performed for a restart following system failure, at most one message per *line* to non-buffered stations, or per process queue for an application program, need be re-sent. If a message was in the process of being sent to a station or application program at the time failure occurred, that message is re-sent automatically if a scan is performed. If a message was in the process of being received from either a station or an application program when failure occurred, that portion of the message that was received and queued before failure occurred is not transmitted following restart, but is lost; the message must be re-entered by the originating station or application program.

When a scan is performed for buffered stations (that is, stations for which the BFDELAY= operand of the TERMINAL macro is coded), at most one message per *station* (that is, the message was in the process of being sent or received when failure occurred) must be retransmitted to make sure no message is lost.

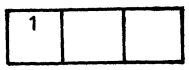
In addition to checking the service bits and eliminating serviced messages from the queues, TCAM also determines whether each message was completely received at the time failure occurred; incomplete messages are purged from the queue and are not sent.

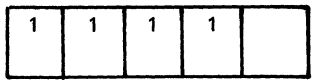
The user may use his *restart in progress* routine to check the input sequence number in the terminal table entry for each station at the time of restart; he might then request by means of a *restart in progress* message that any message entered after the message having this sequence number be re-sent. (Since the input sequence number is not incremented until the entire message has been enqueued, this method will work as long as a SEQUENCE macro instruction is included in the inheader subgroup of the MH handling the message.) The restart in progress routine is described in the discussion of the READY macro instruction in the chapter *Activating and Deactivating the Message Control Program*; no restart in progress facility is available for application programs; suggestions for rendering such programs relatively insensitive to system failure are contained in the section on coordinating TCAM and OS checkpoints in the chapter *Writing TCAM-Compatible Application Programs.*

When no scan is specified for restart following system failure (that is, if WY is coded in the STARTUP= operand of the INTRO macro instruction), upon restart those messages that were on the destination queues waiting to be sent at the time the environment checkpoint being used for restart was taken are sent as if they had been queued just after restart (that is, in FEFO order, according to priority groups). Messages that were on a destination queue waiting to be sent at the time the environment checkpoint was taken (and were subsequently sent before failure occurred) are re-sent following start-up. Messages that were placed on a queue after the environment checkpoint was taken, and were not sent before failure occurred, are not sent after restart; these messages are lost. Incomplete messages are purged from the queues and are not sent. (Incident records are not used so that option fields will reflect the messages that are on each destination queue as a result of using the environment checkpoint record.)

NOTE: If CKPTSYN=YES is specified in a TPROCESS macro instruction, all completely received but unsent messages at the time the last checkpoint request record was made, and all messages completely received between the time the last checkpoint request record was made and the time of failure, are sent upon restart (unless Y is coded in the STARTUP= operand of the INTRO macro instruction; see the description of this operand). See the section on coordinating OS checkpoints with TCAM checkpoints in the chapter *Writing TCAM-Compatible Application Programs* for a discussion of when CKPTSYN=YES would be specified.

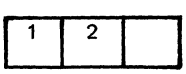
Example:
Consider a checkpoint data set that contains space for three environment checkpoint records and five incident checkpoint records. After the initial environment checkpoint macro executes, assume that four incident checkpoint records are taken. At this point, the condition of the data set can be represented as follows (if we ignore the control record and any checkpoint request records that may be present):

Environment checkpoint records 

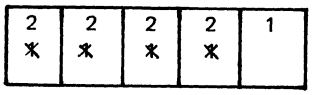
Incident checkpoint records 

Here, a number in the area allotted to an environment record means that the area has been filled with that record; a number in the area allotted to an incident record means that the area is filled with an incident record taken after the environment record having the corresponding number and before an environment record having a higher number.

Assume that another incident checkpoint is taken. This causes the area allotted to incident records to be filled with records taken since the last environment checkpoint; as a result, a second environment record is taken in place of the next incident record. Now, let four more incident checkpoints be taken. These will overlay the earliest incident checkpoints taken after the first environment checkpoint. The data set now has the following appearance (an X over a number means that the record represented by that number has been overlaid):

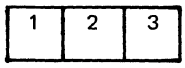
Environment checkpoint records 

Incident checkpoint records

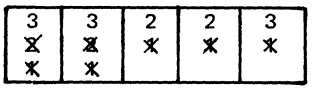


At this point, the time interval specified in the CPINTVL= operand of the INTRO macro instruction expires, resulting in an environment checkpoint, which is followed in turn by three more incident checkpoints. The data set now has the following appearance:

Environment checkpoint records

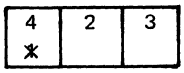


Incident checkpoint records

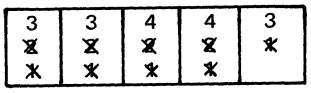


A zone changeover now occurs for a message queues data set on reusable disk, resulting in another environment record, which overlays environment record No. 1. Assume that after two more incident records are taken, system failure occurs. At this point, the checkpoint data set would appear as follows:

Environment checkpoint records



Incident checkpoint records



Assume that the STARTUP= operand of the INTRO macro instruction is coded STARTUP=W, and that the RESTART= operand is omitted. When the restart after failure is performed, the TCAM restart routine will attempt to reconstruct the environment using environment record No. 4 as a base. If record No. 4 is usable, the reconstructed MCP environment created through its use is updated with all the information contained in those incident checkpoint records for which a No. 4 is specified.

In updating, the restart routine begins with the earliest No. 4 incident record, and procedes from earlier to later No. 4 records, continuing to update until all No. 4 incident records have been used; at this point, the MCP environment created by means of environment record No. 4 is considered to be updated.

After the environment record is updated, all message queues in the system are scanned as described above. When updating and scanning are completed, message traffic resumes within one message of the point of failure.

Now, assume that environment record No. 4 is unusable, due, perhaps, to a disk I/O error. In this case, the user would be informed by a message to the system console that the latest environment checkpoint is unusable, and environment record No. 3 would be used as the basis for restart. Environment record No. 3 is updated with all incident records that were taken after it was taken (i.e., those labeled No. 3 and No. 4). In updating, the records still containing information related to incident checkpoint No. 3 are used first, then the two No. 4 records are used, starting with the earlier one; this sequence must be followed in order to ensure that the updated environment record contains the latest available information.

If, after zone changeover, there were three No. 4 incident records rather than two, the earliest No. 3 record would be overlaid. Now, if environment record No. 3 is used as the base, the reconstructed environment probably will not be entirely accurate. This is because the earliest No. 3 record in the incident checkpoint area has been updated for environment record No. 4; when this happened, part of the information related to environment record No. 3 was overlaid. This overlaid information may have referred to environment record fields different from those referred to by the data that overlaid it. The overlaid information was presumably superseded by the information in environment record No. 4, but since the information in environment record No. 4 is inaccessable, the environment record fields that the overlaid information pertains to may contain information that is out of date when restart occurs. Note that this effect is compounded if

environment record No. 3 is also unusable and record No. 2 is used as a base. In the present example, if environment record No. 2 were also inaccessable, the environment could not be reconstructed, since there are no more environment records in the data set. In this case, the system would be started as if no checkpoint records had been taken (i.e., all fields would be initialized to the original values assigned at assembly time), and the checkpoint facility would not be available to the restarted system. The user would get a message at the system console informing him that his environment was not reconstructed, and that his system has no checkpoint facility. To regain his checkpoint facility in this case, the user might close down his system, run the IBCDASDI utility program to assign alternates to defective tracks on the disk containing the checkpoint data set, and then do a cold restart.

How to Get the TCAM Checkpoint Facility

In order to incorporate the TCAM checkpoint facility into his TCAM system, the user must perform the following steps:

1. Include in his MCP a DCB macro instruction defining the checkpoint data set;
2. Include with his MCP a DD statement that allocates space on a disk for the checkpoint data set during initial start-up;
3. Include in his MCP an OPEN macro instruction to open the checkpoint data set.

If the user performs these steps, the TCAM checkpoint facility is included in his TCAM system, where it operates automatically as described in the previous section.

The DCB macro instruction and DD statement for the checkpoint data set are described in the chapter *Defining the MCP Data Sets,* while the chapter *Activation and Deactivation* contains directions for opening the checkpoint data set.

Space must be allocated on disk for the checkpoint data set if the checkpoint facility is desired. The user specifies the number of disk tracks he needs in the SPACE= parameter of the DD statement for the checkpoint data set issued at initial MCP execution time.

Formulas that may be used to determine the number of bytes occupied by the checkpoint data set appear in Figure 34. The formula to be used depends upon whether an IBM 2311 or an IBM 2314 Direct-Access Storage Device is used to contain the data set. Formulas for converting bytes to tracks appear in the component description manual for the direct-access device used.

The formulas in Figure 34 are a bit complex; as an approximate figure, 3 tracks on disk should be sufficient for checkpointing an MCP for which a total of 6 to 10 TERMINAL and TPROCESS macros are coded.

If insufficient storage is allocated for the checkpoint data set, the user may not get as much space allocated for incident records as he wants.

For the IBM 2311 Disk Storage Drive the size in bytes of the checkpoint data set is given by the formula

$$S= (61+1.05L_c)+1.26AL_e+N(61+1.05L_i) + (M+3)\ (61+1.05L_k)$$

For the IBM 2314 Direct Access Storage Device the size in bytes of the checkpoint data set is given by the formula

$$S= (101+1.05L_c)+1.39AL_e+N(101+1.05L_i)+(M+3)\ (101+1.05L_k)$$

In these formulas,

L_c =the length of a control record=30+3A
L_e =the length of an environment record=22+B+C+4D+5E+ (21F1 +21F2 +...+21FE)+(G(H1 +H2 +...+HJ))
L_i =the length of an incident record=12+K
L_k =the length of a checkpoint request record=17+21F+J

Figure 34. Formulas for Determining the Size of the Checkpoint Data Set (Part 1 of 2)

where

A is the value coded in the CPRCDS= operand of the INTRO macro instruction.
B is the total number of bytes of data located in all option fields assigned to stations, lines, or application programs.
C is equal to the sum of the number of single entries in the terminal table plus the number of group entries in the terminal table.
D is equal to the number of single, group, and process entries in the terminal table whose destination queues are maintained on disk.
E is equal to the number of destination queues maintained on disk for single, group, and process entries in the terminal table.
F is equal to the number of priority levels specified for each destination (assume one priority level for each destination queue defined by a TPROCESS macro instruction and one for each destination queue defined by a TERMINAL macro instruction having no LEVEL= operand).
G is equal to 1 if I is specified in the STARTUP= operand of the INTRO macro instruction; otherwise, G is equal to 0.
H is equal to the length of an invitation list (a formula for determining this length is given in the discussion of the ICOPY macro instruction).
I is equal to the number of lines having invitation lists (not counting output-only lines).
J is the length, in bytes, of the maximum number of option fields assigned to any one entry in the terminal table.
K is equal to J if J is greater than 32; otherwise K is equal to 32.
M is equal to the value coded for the CKREQS= operand of the INTRO macro instruction.
N is equal to the number of incident checkpoint records desired (N should be between 1 and 255).

If L_e is less than 300 bytes, it is rounded up to 300 bytes.

Figure 34. Formulas for Determining the Size of the Checkpoint Data Set (Part 2 of 2)

The checkpoint routine uses a priority scheme to divide the space allocated for the checkpoint data set among the various types of checkpoint records. This is to ensure the most efficient use of the checkpoint facility even if less space is provided than would be ideal. Using the available space, the checkpoint facility will:

1. Reserve space for the control record.
2. Reserve space for two environment records.
3. Reserve space for one incident record.
4. Reserve space for the number of checkpoint request records specified in the CKREQ= operand of the INTRO macro instruction, plus 3 (to allow for disk errors).
5. Reserve space for an additional number of environment records sufficient to bring the total number up to that specified in the CPRCDS= operand of the INTRO macro instruction.
6. Use any remaining space to set up additional incident records.

If there is insufficient space for items one through four, the data set is not formatted, no checkpoint facility is provided, and an awareness message is sent to the system console. Also, if there is not enough main storage specified to incorporate either all or a part of the checkpoint/restart facility in the partition or region, the following awareness message is returned to the system console:

IED009I CHECKPOINT DISK ALLOCATION ERROR – DATA SET NOT OPENED

See the OS publication *Messages and Codes* for explanations of and responses to awareness messages.

Types of TCAM Restart

A restart is any TCAM start-up other than the initial startup. A restart may, but need not, involve reconstructing the MCP environment as it existed prior to system closedown or failure.

The three types of restart supported by TCAM are the cold restart, the warm restart, and the continuation restart. These are described below. A cold restart is similar to the initial start-up in that the previous environment is ignored, while the other two types of restart both involve using the TCAM checkpoint facility to reconstruct the environment as it existed before a quick or flush closedown (in the case of a warm restart) or system failure (in the case of a continuation restart).

All three types of restart may be initiated by reloading the object deck for the assembled Message Control Program or by issuing a START command at the system console. For a warm restart or a continuation restart, the DISP= parameter on the DD statement for the checkpoint data set must be coded DISP=OLD; for a cold restart, either DISP=OLD or DISP=NEW may be coded. The chapter *Putting the MCP Together* describes the job control language and procedures for restarting the MCP.

A *cold restart* ignores the previous environment; the system is started as though for the first time. The message queues are considered new and must be reformatted (by means of the IEDQXA routine described in *Appendix E*) before the restart attempt is made (the checkpoint data set, however, is reformatted automatically). A cold restart is performed when the DISP= operand of the DD statement associated with the checkpoint DCB macro is coded DISP=NEW. If the DD statement is coded DISP=OLD, a cold restart is performed following either a quick or a flush closedown if the STARTUP= operand of the INTRO macro instruction has a C coded in it, and is also performed following system failure if the STARTUP= operand of INTRO has CY coded in it. Finally, a cold restart is performed when the TCAM system fails in an attempt to perform a warm or continuation restart because of faulty checkpoint records; in this case, the user is informed by means of a message directed to the system console that a cold restart is being performed. (If none of the environment records can be read at restart time, the checkpoint data set is not opened, so that the ensuing restart is essentially a cold restart.)

If the address of a *good morning* routine is specified in the GMMSG= operand of the READY macro instruction, this routine is given control immediately following a cold restart and before the resumption of normal message traffic. This routine, which is described further in the discussion of the READY macro instruction, may be used to provide specialized initialization for certain stations, and to send a message to each station in the system, informing each that a cold restart has occurred.

There are two forms of restart that reconstruct the environment as it existed before closedown or system failure, the continuation restart and the warm restart. A *continuation restart* involves reestablishing the MCP environment as it existed before system failure. This is done through use of an environment record, incident records, and checkpoint request records in the checkpoint data set, as described above. A continuation restart is performed following system failure if the DISP= operand of the DD statement for the checkpoint data set is coded DISP=OLD, provided that CY is not specified in the STARTUP= operand of the INTRO macro instruction.

During a continuation restart, the message queues may be scanned (as described in the previous section) to determine the last complete message received and transmitted before failure for each queue; whether scanning is performed depends upon how the STARTUP= operand of the INTRO macro instruction is coded (see the description of this operand). If synchronization with OS checkpoints of an application program is specified for a particular process queue by coding CKPTSYN=YES in its TPROCESS macro instruction, that queue is scanned during restart; upon restart, those complete messages that were marked serviced after the last checkpoint request record was made, or were enqueued after this record was made, are sent. (If no checkpoint request records were made between the time of start-up and the time of failure, all messages marked serviced or received since the last environment checkpoint record was made are sent upon restart).

For a *warm restart* following a quick or flush closedown, the MCP environment is reconstructed as for a continuation restart. Since an environment checkpoint is taken near the end of a quick or flush closedown, no incident records need be used to reconstruct the MCP environment during a warm restart. A warm restart is performed if the DISP= operand of the DD statement for the checkpoint data set is coded DISP=OLD, provided that a W or WY is coded in the STARTUP= operand of the INTRO macro instruction and that the restart follows a quick or flush close (both of which are described in the *Deactivation* section of the chapter *Activation and Deactivation*).

If the address of a *restart in progress* routine is provided in the RSMSG= operand of the READY macro instruction, this routine is given control immediately following a warm restart or continuation restart. This routine, which is described further in the discussion of the READY macro instruction, may be used to gain access to and to change option fields and information contained in terminal table entries, and may be used to inform each station that a warm or continuation restart has occurred. The message might also provide each station with the input sequence number of the last message received from the station, and request that all messages entered having higher sequence numbers be reentered.

Below is a summary of the conditions that must be met in order to obtain each of the types of restart described in this section. It is assumed that the DISP= operand of the DD statement associated with the checkpoint DCB macro instruction is coded DISP=OLD; if DISP=NEW is coded, a cold restart is always performed.

Type of Termination	*INTRO Operand STARTUP=*	*Resulting Restart*
Flush closedown	W or WY	Warm restart
Flush closedown	C or CY	Cold restart
Quick closedown	W or WY	Warm restart
Quick closedown	C or CY	Cold restart
System failure	C or W	Continuation restart with queue scan
System failure	WY	Continuation restart with no queue scan
System failure	CY	Cold restart

The user may wish to specify a warm restart following a flush closedown in order to avoid the loss of messages that could not be flushed during closedown either because an application program was closed or because a station was inoperative or intercepted.

Using TCAM's Message Logging Facility

TCAM's message logging facility enables the user to keep a record of the message traffic handled by an MCP on a sequential data set. The LOG macro instruction causes either a message or a message segment to be recorded on a log data set while the message is currently being processed by an MCP subgroup.

Uses of Message Logging

Message logging can be useful to the programmer of a telecommunications system in two ways: first, as an integral part of the system, recording messages for accounting purposes by the user, and second, as a programming aid, helping to diagnose errors and providing information needed to evaluate system performance.

Message Logging as a System Component: In some systems, it may be desirable for messages to be recorded for accounting purposes, even though the messages have been successfully dispatched to their destinations. This allows the programmer greater flexibility in his accounting procedures. Some uses of a logging facility might be:

- copying groups of messages sent over a long period of time to a variety of destinations,
- providing long-term back-up for messages that might be accepted by one or more destinations but later lost through human error, and
- enabling collection of exceptional cases.

Message logging can provide any of these functions without requiring that an application program be written.

Message Logging as a Programming Aid: Including a carefully designed message logging facility in a Message Handler permits the programmer to trace the flow of messages through a Message Control Program, thus allowing quick diagnosis of errors while debugging the MCP. By anticipating the need for debugging aids in the design of his

message logging facility, the programmer can provide a useful diagnostic tool with very little programming effort. Because of its modular design, the message logging facility can be removed easily, without the necessity of rewriting any parts of the MCP involved, when the program is free of errors.

By determining the flow patterns of message traffic, a programmer can more efficiently allocate the resources of a telecommunications system. Message logging assists the programmer as a data collection facility, providing the information needed to make such a determination. When the TCAM MCP first executes, it can include the code necessary to log information such as time, origin, and destination for each message, or in cases where traffic is heavy, for certain representative messages. The programmer is then able to reallocate resources efficiently, and he can easily remove the message logging facility when it is no longer needed.

A later section titled *Debugging Aids* includes message logging among the various techniques that might be used to aid in debugging the TCAM environment.

How Message Logging Works

When a LOG macro executes in an MH subgroup, either a complete message or a message segment is copied as it then exists onto a log data set. The operand coded on the LOG macro and the type of subgroup in which the LOG macro appears determine what is to be logged – message segment or complete message. If only a segment is to be logged, an operand of the LOG macro refers directly to the DCB for the log data set. If an entire message is to be logged, the operand refers to a LOGTYPE macro that points to the DCB and contains additional information necessary to log multiple segments. The relationship of the LOG macro and the various subgroups is discussed in *What to Log.* Figure 35 shows the flow of data and control that occurs during the logging process for message segments and complete messages.

How to Set Up a Message Logging Facility

This section discusses the elements of a message logging facility in the order in which the programmer is likely to deal with them while writing an MCP. Complete descriptions of the macro instructions discussed in this section (LOG, LOGTYPE, PATH, MSGTYPE, and the log DCB macros) may be found elsewhere in this publication.

What to Log The logging facility can handle the following:

- incoming header segments,
- incoming segments,
- incoming messages,
- outgoing header segments,
- outgoing segments, and
- outgoing messages.

When a LOG macro executes, the message unit (either a complete message or a message segment) that is being processed by its subgroup is logged. The type of unit to be logged is determined by the type of subgroup in which the LOG appears; if it is coded in an inheader or outheader subgroup, message segments containing message headers are logged. Its occurrence in an inbuffer or outbuffer subgroup causes each segment to be logged. Each complete message is logged as a unit when the LOG macro occurs in an inmessage or outmessage subgroup.

As larger units are logged, the logging facility requires more processing time and main storage space, and the logged messages occupy more space in the log data set. Therefore, the message logging facility should be designed to operate on the smallest units that will supply the information needed. In a message switching application, simply logging ncoming headers may supply all the information that is needed for a logging application. ı more sophisticated applications where the body of a message is needed, it may be ·cessary to log complete messages.

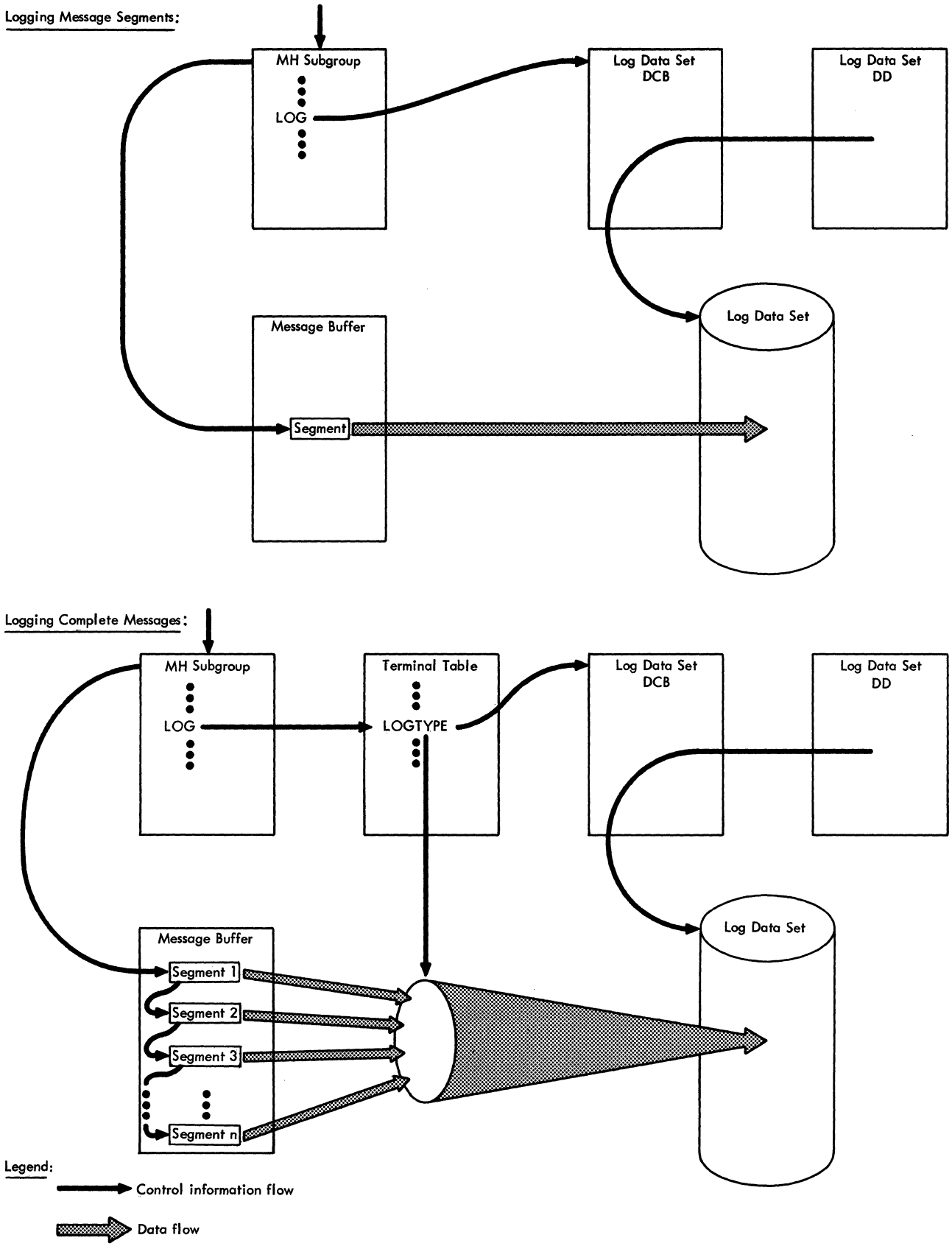


Figure 35. Information Flow for Message Logging.

The Log Data Set and its DCB: Logged messages and message segments are maintained on sequential data sets residing on any type of BSAM-supported device. There may be any number of log data sets for any given MCP. Multiple log data sets may be found useful where logged messages are differentiated by some program-discernable characteristic, such as format, destination, or source. The device upon which a log data set resides must be able to handle the volume of information expected to be logged at least as fast as the MCP can handle it. This prevents a backlog of messages from accumulating and being lost as buffer areas fill and are overwritten with new messages. Some devices that might be used for logging purposes are tape, disk, and, where traffic is light, printer.

The DCB for a log data set is coded with the rest of the DCBs for the MCP. Coding details appear in the section *Log Data Sets* in the chapter *Defining MCP Data Sets.* Note that the size of the records is determined by the size of the buffer units used in the MCP.

The LOGTYPE Macro: The LOGTYPE macro provides the additional information needed by a message logging facility when it is to log complete messages (that is, when a LOG macro appears in an inmessage or outmessage subgroup). Since messages consist of a series of message segments, buffer and queue areas must be defined; the BUFSIZE= and QUEUES= operands of the LOGTYPE macro are used for this purpose. When a LOGTYPE macro is needed, it should appear in the terminal table section of the MCP. Coding details appear in the section *LOGTYPE Macro Instruction* in the chapter *Defining Terminal and Line Control Areas.*

The LOG Macro: The LOG macro causes a message or message segment to be logged When it is encountered in an MH subgroup, the currently processed unit is transferred to the logging medium or to a queue waiting for such a transfer. The operand of the macro refers either to the log data set or the LOGTYPE macro associated with this particular LOG macro. Coding details for the LOG macro are discussed in the section *LOG Macro Instruction* in the chapter *Designing the Message Handler.*

Selective Logging: It may be desirable to log only messages that meet certain criteria, instead of each message handled by a particular MH subgroup. Use of the PATH and MSGTYPE macros enables the programmer to include decision-making code in the message logging facility. The chapter *Designing the Message Handler* contains discussions of facilities provided by both of these macro instructions.

Debugging Aids

During the execution of a TCAM MCP, error messages may be directed to the system console and to operator control stations. Each TCAM message starts with an alphameric identifier; an exact definition of the message and any user action that may be required is documented in alphameric order in *Messages and Codes* (this document also lists and defines codes).

When the MCP partition or region is dumped, the MCP control blocks are formatted (described in the *TCAM PLM*). Among the blocks formatted are the terminal table, station control blocks, line control blocks, queue control blocks, data control blocks, process control blocks, and the address vector table.

In addition to normal dumps of the MCP partition or region, TCAM provides some special aids for debugging the telecommunications network and the MCP. Two of these have already been described in this chapter; the I/O error recording facility, described in the section *TCAM I/O Error-Recording Procedures,* and the TCAM logging facility covered in the section *Using TCAM's Message Logging Facility.* A TCAM formatted ABEND dump is taken of the TCAM MCP partition or region that terminates abnormally; this ABEND dump, which is in addition to the OS ABEND dump, formats TCAM control areas and attached subtasks and is discussed in the *TCAM PLM.* Other optional debugging aids include a cross-reference table of line-related information, located in main storage, and special dumps of a subtask control block (STCB) trace, line I/O interrupt trace for a line, buffers, and message queues data sets. In addition, the cross-reference table, STCB trace, and line I/O interrupt trace may reside in main storage and may be included in a standard dump.

Figure 36 at the end of this section lists coding requirements for using the special TCAM debugging aids, and the sections below discuss the individual aids. The *Diagnostic Aids* section of the *TCAM PLM* contains several tables that should be useful in debugging a TCAM system.

Cross-Reference Table

The TCAM cross-reference table provides the user with a convenient means of locating in a standard OS dump certain information associated with each open line. The cross-reference table is built by TCAM if the user codes a non-zero integer in the CROSSRF= operand of the INTRO macro instruction.

At INTRO execution time, TCAM allocates 16n+8 contiguous bytes of main storage (where *n* is the integer specified in the CROSSRF= operand and 8 bytes is the length of the control block preceding the first entry) for the cross-reference table, and places the address of the cross-reference table in the AVT (address vector table) field labeled AVTCRSRF. Each time a line is opened, the next available four-word entry in the cross-reference table is filled in for that line.

The format of the 8-byte control block preceding the first entry is:

Byte	*Explanation*
0	address of first available entry
+4	address of last entry

The format of each entry in the table is:

Byte	*Explanation*
0	unit control block name
+4	unit control block address
+8	line control block address
+12	address of a master queue control block for this line

If queuing is by line, there is only one master queue control block assigned to the line, and its address is placed in the fourth word. If queuing is by terminal, there is a master queue control block for each station on the line; the fourth word in this instance is filled in with the address of the queue control block for the station whose terminal table entry appears in the terminal table before that of any other station on the line. (The line control block and the queue control block are internal TCAM control blocks and are discussed in the *TCAM Program Logic Manual.*) If the user opens more lines than he provides entries for in the cross-reference table, the table is filled in until the space in it is exhausted; lines opened after space runs out in the table have no cross-reference entries.

TCAM Line I/O Interrupt Trace Table

The TCAM line I/O interrupt trace table provides a sequential record (referred to as a line I/O trace) in main storage of the I/O interrupts occuring on a specified line. When an I/O interrupt occurs on a line for which a line I/O trace is requested (by the GOTRACE operator command), information about the interrupt, including the CSW and the CCW, is stored as an entry in the line I/O interrupt trace table; however, interrupts resulting from retries by TCAM's error recovery procedures are not recorded.

The line I/O interrupt trace facility is brought into main storage by specifying a positive value (from 1 to 65535) in the TRACE= operand of the INTRO macro instruction; once it is in main storage, it may be activated and deactivated for a specified line by the GOTRACE and NOTRACE operator commands, respectively.

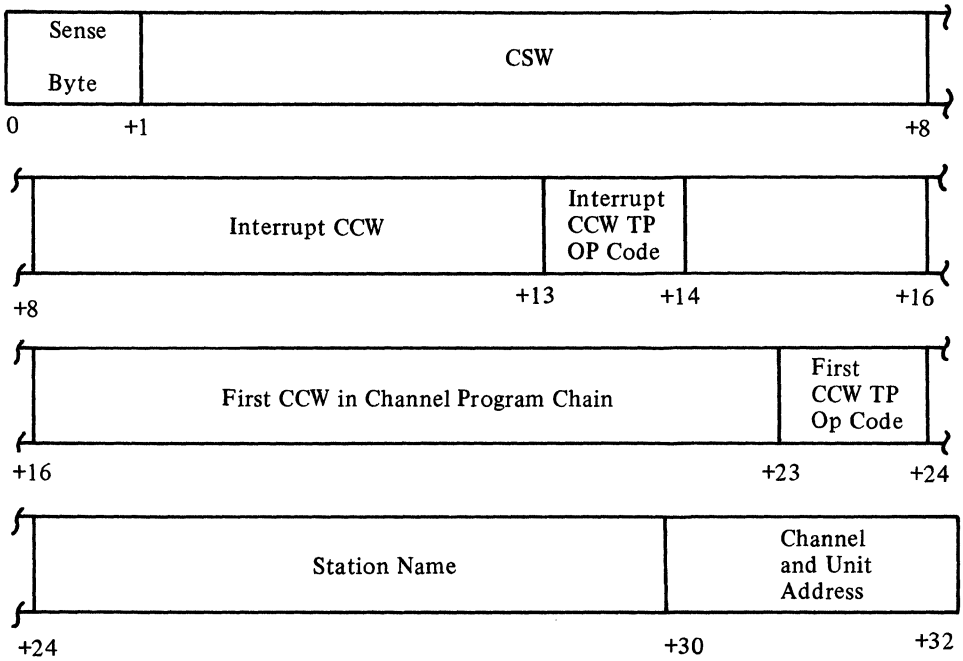
At INTRO execution time, 32n+16 contiguous bytes of main storage (where *n* is the integer specified in the TRACE= operand and 16 bytes is the length of the control block preceding the first entry) are allocated for the line I/O interrupt trace table; TCAM places the address of the table in the AVTRACE field of the AVT. When all the 32-byte entries in the I/O trace table have been filled, the earliest entries are overlaid as new interrupts occur.

A standard OS dump, described in the *Programmer's Guide to Debugging,* may be obtained to determine the kinds of interrupts that occurred during execution of the MCP.

The format of the 16-byte control block preceding the first entry is:

Byte	*Explanation*
0	address of current trace entry
+4	address of first trace entry
+8	address of last trace entry
+12	address of middle entry

Each entry in the TCAM I/O interrupt trace table has the following format:



The channel programs used by TCAM may be found in the *TCAM Program Logic Manual.* A teleprocessing operation code (TP Op code) is assigned to each CCW, and may be found in the sixth byte of each CCW in the table. TP Op codes facilitate a trace of the channel program execution sequence – they are described in the *TCAM Program Logic Manual.* A detailed description of the contents of the sense byte may be found in the component description SRL publication for the transmission control unit being used. If the identity of the connected station is not known when the interrupt occurs, then the channel and unit addresses, in unpacked form, are placed in the last two bytes of the 32-byte entry.

Writing on a Data Set for Later Printing: The user invokes the I/O trace dump (IEDQFE20) by entering the DEBUG operator command; this routine requires that COMWRTE=YES be specified on the INTRO macro instruction.

As soon as TCAM makes half the number of entries in the line I/O interrupt trace table that was specified on the TRACE= operand of the INTRO macro instruction, the I/O trace dump routine passes that portion of the trace table to the COMWRITE routine to be written on a sequential data set (the requirements of COMWRITE are explained in the next section below). TCAM makes entries in the second half of the trace table until that section is filled, at which time the I/O trace dump routine again passes data to COMWRITE. TCAM continues to overlay the line I/O interrupt trace table, and the process outlined above is repeated.

COMWRITE Requirements and Format: The output data set format is undefined with a maximum block size that is permissible for the particular device. An example of the JCL required to specify that the output data set (COMWRITE) be on tape is:

```
//COMWRITE DD UNIT=2400,DSN=COMWRITE,VOL=SER=xxxxxx,DISP=(,KEEP)
```

Multiple volumes (either labeled or unlabeled) may be specified; secondary allocation is not permitted. Once a disk data set is filled, the disk is wrapped with subsequent entries.

Sample JCL for specifying that the output data set be on disk is:

```
//COMWRITE  DD  UNIT=SYSDA,DSN=COMWRITE,DISP= (,KEEP),          X
//                VOL=SER=xxxxxx,SPACE=(CYL,(5))
```

The COMWRITE data set is used also when sequentially writing the STCB trace and buffers (the data set is specified only once); the appropriate operand of the DEBUG operator command determines whether a line trace, an STCB trace, or buffers are written on the data set (see examples in *Writing Line Trace, STCB Trace, and Buffers to Disk Data Set* below). Output from the COMWRITE data set is printed by a separate task that is discussed below in *COMEDIT Printing Utility*.

If too few entries are specified on the TRACE= operand of the INTRO macro instruction, COMWRITE may become too busy to forward all records to the data set; such records are lost and the I/O trace dump routine reuses that section of the table and the count field of the output indicates a missing record. When the printed output indicates lost records, increase the number of entries on the TRACE= operand of the INTRO macro to prevent reoccurrence.

NOTE: Since the blocksize is limited by the type of storage device, care must be taken in defining the sizes of the various main storage trace tables to be recorded by COMWRITE. Since each record is one-half the trace table, no trace table can exceed twice the maximum blocksize permitted for the COMWRITE external storage device. This implies the need for tape to record extra large main storage tables since tape supports larger records.

Dispatcher Subtask Trace Table

The dispatcher subtask trace table is used to keep a sequential record in main storage of the subtasks activated by the TCAM dispatcher. An entry is placed in the table by the TCAM dispatcher each time a TCAM subtask is dispatched. The table is filled on the wraparound principle; that is, when all of the available entries have been used, the dispatcher places the new entries at the beginning of the table thus overlaying the earliest entries. The table might be used, for example, to trace the path of a buffer through the TCAM system. The TCAM dispatcher and TCAM subtasks are described in detail in the *TCAM Program Logic Manual.*

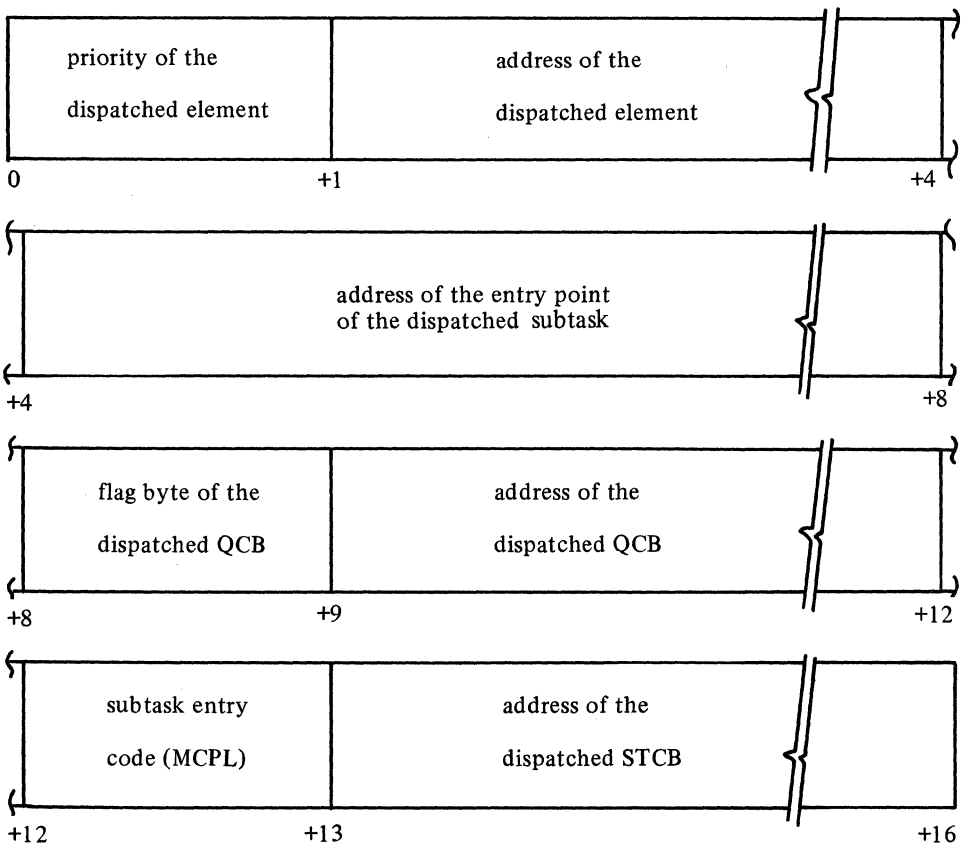
The dispatcher subtask trace table is generated if the user specifies some value between 4 and 65,535 in the DTRACE= operand of the INTRO macro (see note below). At INTRO execution time, TCAM allocates (16n + 16) contiguous bytes of main storage (where *n* is the integer specified in the DTRACE= operand) for the table. The table consists of a 16-byte control block followed by *n* 16-byte entries. The address of the 16-byte control block is then stored in a field in the AVT (address vector table) at a displacement of 12 bytes past the label IEDPARM.

The control block for the dispatcher subtask trace table has the following format:

Byte	*Explanation*
0	Address of next entry to appear in the table
+4	Address of first entry in the table
+8	Address of last entry in the table
+12	Size of the table in bytes

NOTE: Since the first word of this control block contains the address of the next entry in the table, the last recorded entry is located at this address minus 16.

An entry in the dispatcher subtask trace table has the following format:



For a further description of the QCB, the QCB flag byte, the STCB, the subtask entry code (MCPL), and the priority of the dispatched element, refer to the *TCAM Program Logic Manual.*

The subtask trace, like the line trace, can be written sequentially to a data set (either magnetic tape or disk) provided the COMWRTE= operand of the INTRO macro instruction specifies YES. See the DEBUG operator command for a description of how to activate and deactivate the STCB trace dump (IEDQFE10). The requirements of COMWRITE are discussed earlier in *COMWRITE Requirements and Format.*

NOTE 1: As soon as TCAM has made half the number of entries in the subtask trace table that is specified on the DTRACE= operand of the INTRO macro instruction, IEDQFE10 determines that new entries exist in the first half of the table and passes the first half of the trace table to COMWRITE to be written on a sequential data set. TCAM continues, without interruption, filling in the last half of the table with entries. When the second half is filled, IEDQFE10 passes the second half to COMWRITE to be written. Each time IEDQFE10 is entered, a counter is incremented and placed in the output record.

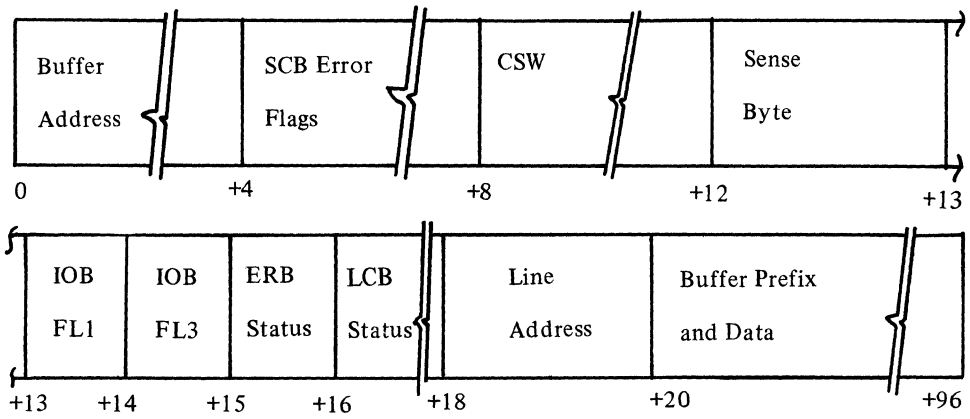
If too few entries are specified on the DTRACE= operand of the INTRO macro instruction, COMWRITE may become too busy to forward all records to the data set; such a records are lost and IEDQFE10 reuses that section of the table and the count field of the output indicates a missing record. When the printed output indicates lost records, increase the number of entries on the DTRACE= operand of INTRO to prevent reoccurrence.

NOTE 2: Since the blocksize is limited by the type of storage device, care must be taken in defining the sizes of the various main storage trace tables to be recorded by COMWRITE. Since each record is one half the trace table, no trace table can exceed twice the maximum blocksize permitted for the COMWRITE external storage device. This implies the need for tape to record extra large main storage tables since tape supports larger records.

Buffer Dump

This facility permits TCAM buffer contents and status information to be dumped to a data set residing either on magnetic tape or on disk. To get this facility, the user specifies YES on the COMWRTE= operand of the INTRO macro instruction (causing a routine to be attached to the TCAM partition or region that sequentially writes buffers to tape or disk), and enters the DEBUG operator command to activate IEDQFE30. The data set to which buffers are written is specified by the JCL that is discussed above in *COMWRITE Requirements and Format.* The buffer dump also requires that a line trace be active on the line whose buffers are to be dumped (see *I/O Interrupt Trace Table*, covered earlier in this section, for a discussion of how to activate the line trace).

The format of the buffer dump on tape or disk is:



See a later section, *COMEDIT Printing Utility* for a discussion of how the COMEDIT routine can produce a formatted listing of the buffer.

Writing Line Trace, STCB Trace, and Buffers to Disk Data Set

The following example of entering GOTRACE, NOTRACE, and DEBUG operator commands illustrates how debugging information can be written to the COMWRITE data set for later printing. The general format of the commands and command responses used in the example may be found in this chapter under the heading *Operator Commands.* The

numbered paragraphs below denote a sequence of operator commands entered at an IBM 1050 Data Communication System terminal that has been designated as an operator control station. The lettered entries designate discussions of

A. STCB trace,
B. line I/O trace, and
C. buffer dump, respectively.

The example assumes that the INTRO macro instruction specifies COMWRTE=YES, DTRACE=integer, TRACE=integer, and CONTROL=CTL; the jobname of the TCAM MCP is TEST3; the following DD statement has been added to the EXEC statement for the TCAM MCP

```
//COMWRITE   DD    UNIT=SYSDA,DSN=COMWRITE,DISP= (,KEEP),      X
//                 VOL=SER=456789,SPACE=(CYL,(5))
```

(specifies that the COMWRITE data set be on disk); and the TCAM job is dequeued from the input stream and has just started execution.

Example:

1. Nothing has been entered at the IBM 1050 terminal.
   A. Entries are being made in the STCB trace table in main storage, and nothing is being written to the COMWRITE data set.
   B. Inactive.
   C. Inactive.
2. CTL F TEST3,TRACE=052,ON (eot)
   A. Entries are still being made in the STCB trace table, and nothing is being written to the COMWRITE data set.
   B. Entries are now being made in the line I/O trace table in main storage for line 052, and nothing is being written to the COMWRITE data set. Response message

      IED023I TRACE STARTED FOR 052

      is sent to the IBM 1050 indicating that the line I/O trace was started for line 052 as requested by the GOTRACE command.
   C. Inactive.
3. CTL F TEST3,DEBUG=L,IEDQFE10 (eot)
   A. Entries are still being made in the STCB trace table, and now the STCB trace table entries are being written to the COMWRITE data set. Response message

      IED099I ROUTINE LOADED

      is sent to the IBM 1050 indicating that the STCB trace table entries are being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for line 052, and nothing is being written to the COMWRITE data set.
   C. Inactive.
4. CTL F TEST3,DEBUG=L,IEDQFE30 (eot)
   A. Entries are still being made in the STCB trace table, and STCB trace table entries are still being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for line 052, and nothing is being written to the COMWRITE data set.
   C. Since line I/O trace is active for line 052, buffer and status information for line 052 is now being written to the COMWRITE data set. Response message

      IED099I ROUTINE LOADED

      is sent to the IBM 1050 indicating that buffer and status information is being written to the COMWRITE data set.

5. CTL F TEST3,TRACE=031,ON (eot)
   A. Entries are still being made in the STCB trace table, and STCB trace table entries are still being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for line 052, entries are also being made now in the line I/O trace table in main storage for line 031, and nothing is being written to the COMWRITE data set. Response message

      IED023I TRACE STARTED FOR 031

      is sent to the IBM 1050 indicating that the line I/O trace was started as requested by the GOTRACE command.
   C. Since line I/O trace is active for lines 052 and 031, buffer and status information for these lines is being written to the COMWRITE data set.
6. CTL F TEST3,DEBUG=L,IEDQFE20 (eot)
   A. Entries are still being made in the STCB trace table, and STCB trace table entries are still being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for lines 052 and 031, and now line I/O trace table entries are being written to the COMWRITE data set. Response message

      IED099I ROUTINE LOADED

      is sent to the IBM 1050 indicating that line I/O trace table entries are now being written to the COMWRITE data set.
   C. Since line I/O trace is still active for lines 052 and 031, buffer and status information for these lines is still being written to the COMWRITE data set.
7. CTL F TEST3,TRACE=052,OFF (eot)
   A. Entries are still being made in the STCB trace table, and STCB trace table entries are still being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for line 031, but not line 052, and line I/O trace table entries for line 031 are being written to the COMWRITE data set. Response message

      IED029I TRACE STOPPED FOR 052

      is sent to the 1050 indicating that the line I/O trace was stopped for line 052 as requested by the NOTRACE command.
   C. Since line I/O trace is now active only for line 031, buffer and status information for line 031 is still being written to the COMWRITE data set.
8. CTL F TEST3,DEBUG=D,IEDQFE30 (eot)
   A. Entries are still being made in the STCB trace table, and STCB trace table entries are still being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for line 031, and line I/O trace table entries for line 031 are still being written to the COMWRITE data set.
   C. Inactive.
9. CTL F TEST3,DEBUG=D,IEDQFE20 (eot)
   A. Entries are still being made in the STCB trace table, and STCB trace table entries are still being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for line 031, but now nothing is being written to the COMWRITE data set. Response message

      IED100I ROUTINE DEACTIVATED

      is sent to the IBM 1050 indicating that no line I/O trace table entries are being written to the COMWRITE data set.
   C. Inactive.
10. CTL F TEST3,DEBUG=D,IEDQFE10(eot)
   A. Entries are still being made in the STCB trace table, but now nothing is being written to the COMWRITE data set. Response message

      IED100I ROUTINE DEACTIVATED

      is sent to the IBM 1050 indicating that no STCB trace table entries are being written to the COMWRITE data set.
   B. Entries are still being made in the line I/O trace table for line 031, and nothing is being written to the COMWRITE data set.
   C. Inactive.

11. CTL F TEST3,TRACE=031,OFF (eot)
   A. Entries are still being made in the STCB trace table, and nothing is being written to the COMWRITE data set.
   B. Inactive. Response message

      IED029I TRACE STOPPED FOR 031

      is sent to the IBM 1050 indicating that entries are no longer being made in the line I/O trace table.
   C. Inactive.

COMEDIT Printing Utility

The COMEDIT utility is a separate job or job step that formats and prints in hexadecimal the specified output from COMWRITE data set. The COMWRITE data set may contain any combination of entries from the STCB trace table, the I/O interrupt trace table, and buffer and status information. If the COMWRITE data set is on magnetic tape, a search option may be invoked to begin formatting the STCB trace dump at a specified time stamp (if the search option is requested and the COMWRITE data set is on disk, error message IED120I 'BLOCK=' PARM REQUIREDTAPE INPUT is returned to the system console and the printing utility formats and prints all the data from the physical beginning of the COMWRITE data set). The number of lines printed per page on the listing may be varied if the default value of 60 is not desired. Options that may be specified on the PARM= parameter of the EXEC statement are:

Option	*Function Provided*
STCB	Provides a formatted printout of the dispatcher subtask trace table entries that are on the data set.
IOTR	Provides a formatted printout of the line I/O interrupt trace table entries that are on the data set.
BUFF	Provides a hexadecimal and EBCDIC formatted printout of buffers and any trace records, other than STCB and IOTR, that are on the data set.

NOTE: If neither STCB, IOTR, nor BUFF is specified, then all blocks are formatted and the hexadecimal dump contains all unknown records.

BLOCK=hhmmddd	This keyword value is valid only for a tape data set; it designates the starting point for formatting records from the data set. *hh* is replaced by a 2-digit decimal integer that specifies the hour in continental time. *mm* is replaced by a 2-digit decimal integer that specifies the minutes of the hour (using 1-minute intervals). *ddd* is replaced by a 3-digit decimal integer that specifies the day of the year (using Julian days).
LINECNT=xx	Specifies the number of lines to be printed per page, where *xx* may be replaced by any 2-digit decimal integer up to 99. If this keyword parameter is omitted, a value of 60 is assumed.

These options may be coded in any order and as often as desired. If keyword parameters are specified more than once, only the last duplicate parameter is honored. If a parameter is coded incorrectly, the entire parameter list is printed on the SYSPRINT data set and the location of the parm scan pointer is shown, indicating the approximate location of the erroneous parameter.

Examples appear below for invoking various combinations of entries from the COMWRITE data set; the ddnames SYSPRINT and SYSUT1 are required names for the DD statements in this program.

The DCB attributes for SYSPRINT are

DSORG=PS,RECFM=FBA,LRECL=121,BLKSIZE=1331.

Example 1:
Prints only STCB trace table entries from the disk COMWRITE dat set, and specifies a line count of 80 for the printed listing.

```
//jobname    JOB
//stepname   EXEC     PGM=IEDQXB,PARM='LINECNT=80,STCB'
//SYSPRINT   DD   SYSOUT=A
//SYSUT1     DD   DSN=COMWRITE,UNIT=SYSDA,VOL=SER=xxxxxx,DISP=OLD
```

Example 2:
Prints STCB trace table entries and line I/O interrupt trace table entries from tape; specifies printed listing to begin at 10:15 pm on August 20, 1970, and specifies line count of 58.

```
//jobname    JOB
//stepname   EXEC     PGM=IEDQXB,PARM= 'IOTR,BLOCK=2215232,      X
//               STCB,LINECNT=58'
//SYSPRINT   DD   UNIT=00E
//SYSUT1     DD   UNIT=2400,VOL=SER=xxxxxx,DISP=OLD,LABEL= (,NI)
```

NOTE: If the search option is earlier than the first time stamp on the data set containing the STCB trace table entries, the entire data set is formatted and printed. If it is later than the last time stamp, no printing is done and error message IED121I REQUESTED TIME NOT FOUND is returned. In addition to the STCB trace being formatted by time and date, queue control blocks are formatted by individually named fields. I/O interrupt trace table entries on the COMWRITE data set appear as 32-byte records with unit check and unit exception fields flagged.

Example 3:
Prints buffers from tape – standard label in; standard label out (for printing on another machine).

```
//jobname    JOB
//stepname   EXEC     PGM=IEDQXB,PARM='BUFF'
//SYSPRINT   DD   DSN=dsname,UNIT=2400,VOL=SER=xxxxxx,DISP= (,KEEP)
//SYSUT1     DD   UNIT=2400,VOL=SER=xxxxxx,DISP=OLD,DSN=COMWRITE
```

Example 4: Prints buffer and status information and line I/O interrupt trace table entries from tape; specifies line count of 65 (if duplicate keyword parameters are specified, the later value is used).

```
//jobname    JOB
//stepname   EXEC     PGM=IEDQXB,PARM='LINECNT=10,BUFF,IOTR, X
//               LINECNT=65'
//SYSPRINT   DD   SYSOUT=A
//SYSUT1     DD   UNIT=2400,VOL=SER=TRACE,DISP=OLD,LABEL= (,NL)
```

NOTE: If the PARM= parameter is omitted, all the entries on the COMWRITE data set are formatted and printed. If a coding error is detected in one of the keyword values of the PARM= parameter, error message IED123I INVALID PARAMETERS is returned indicating that the COMEDIT printing utility cannot be continued due to invalid JCL: replace the invalid JCL card and resubmit the job.

Message Queues Data Set Dump

TCAM provides a separate utility (IEDQXC) that formats the DASD message queues data set for immediate printing, or it directs the messages queues data set to either magnetic tape or disk for later printing. (See the discussion on *Preformatting DASD Message Queues Data Sets* in the chapter titled *System Preparation*). The entire data set may be printed sequentially either by record number or by queue. Also, up to five individual queues may be printed. The contents of the formatted dump are controlled by options in the PARM= parameter of the EXEC statement. The general format of the EXEC statement is:

```
//STEP1          EXEC PGM=IEDQXC,PARM='Q=options'
```

Options that may be specified on the PARM= parameter of the EXEC statement are:

Option	*Function Provided*
DMP	Prints all messages sequentially by record number.
xxx,DMP	Prints all messages sequentially, where *xxx* is replaced by the 3-digit decimal total number of queues.

NOTE 1: Specifying *xxx*,DMP gives the same results as either specifying DMP alone or omitting the PARM= parameter.

NOTE 2: To find *total number of queues* from the assembly listing of the MCP, look for ORG IEDQNADDR in the expansion of the macro that is named by TTABLE LAST=*name*. Following that ORG is the line DC A(*n* + 1),A(*r* + 1). The value of *n* (for nonreusable disk) or *r* (for reusable disk) is the maximum value for Q=*xxx* in the PARM= parameter of the IEDQXB routine.

xxx,ALL	Prints all messages sequentially by queue, where *xxx* is replaced by the 3-digit decimal total number of queues.
xxx,$n_1n_1n_1$,$n_2n_2n_2$, ...$n_5n_5n_5$	Prints all messages for queues $n_1n_1n_1$ through $n_5n_5n_5$ (up to 5 queues may be specified); *xxx* is replaced by the 3-digit decimal total number of queues, and *nnn* is replaced by the 3-digit decimal number corresponding to the queue (or queues) whose contents are to appear in the dump.

Below are some sample JCL statements for invoking the IEDQXC printing facility.

NOTE 3: Each extent of the DASD message queues data set must be defined with a DISKQnn card, where *nn* is replaced by decimal 01 for the first extent, by decimal 02 for the second, etc. For single extent cataloged data sets, DSN= and DISP= are the only required parameters. For multi-extent (multivolume) data sets, the catalog information cannot be used. Each DD statement must have also UNIT=23xx,VOL=SER=xxxxxx information. These DD statements must define the volume identification in the same order as the volume identifications listed in the IEDQDATA DD card on the IEDQXA utility JCL used when creating the data set.

Example 1:
Dumping queues sequentially

```
//jobname          JOB
//stepname         EXEC PGM=IEDQXC
//DISKQ01    DD    DSN=dsname,DISP=OLD
//SYSPRINT   DD    UNIT=00F
```

Example 2:
Printing entire formatted queue

```
//jobname          JOB
//stepname         EXEC PGM=IEDQXC,PARM='Q=012,ALL'
//DISKQ01    DD    DSN=dsname,DISP=OLD,UNIT=23xx,VOL=SER=111111
//DISKQ02    DD    DSN=dsname,DISP=OLD,UNIT=23xx,VOL=SER=222222
//DISKQ03    DD    DSN=dsname,DISP=OLD,UNIT=23xx,VOL=SER=333333
//SYSPRINT   DD    UNIT=00E
```

Example 3:
Printing selected queues (there are eight queues on the data set; this JCL formats and prints queues 005,006, and 007 only)

```
//jobname          JOB
//stepname         EXEC PGM=IEDQXC,PARM='Q=008,005,006,007'
//DISKQ01    DD    DSN=dsname,DISP=OLD,UNIT=23xx,VLL=SER=111111
//DISKQ02    DD    DSN=dsname,DISP=OLD,UNIT=23xx,VOL=SER=222222
//DISKQ03    DD    DSN=dsname,DISP=OLD,UNIT=23xx,VOL=SER=333333
//SYSPRINT   DD    UNIT=00E
```

The PARM= fields are fixed format, and a coding error causes error message IED123I INVALID PARAMETERS to be returned to the system console; replace the invalid JCL and resubmit the job.

Debugging Aid ↓	Function				
	Getting debugging aid to reside in main storage		Dumping debugging aid to either tape or disk data set		Printing debugging aid from tape or disk data set
	INTRO operands	operator commands	INTRO operands	operator commands	
Standard OS Dump		(Note 1)			
TCAM Formatted ABEND Dump		(Note 2)			
I/O Error Recording		(Note 3)			See the IFCEREPO system utility in the OS Utilities Publication for printing the contents of the SYS1. LOGREC data set.
TCAM Logging		(Note 4)			
Cross-Reference Table	CROSSRF=n				
STCB Trace	DTRACE=n		DTRACE=n COMWRTE=YES	DEBUG (L, IEDQFE10)	
Line Trace	TRACE=n	GOTRACE NOTRACE	TRACE=n COMWRTE=YES	GOTRACE DEBUG (L, IEDQFE30)	Use the COMEDIT printing utility described in this section.
Buffer Dump			TRACE=n COMWRTE=YES	GOTRACE DEBUG (L, IEDQFE20)	
Message Queues Data Set Dump					Use the IEDQXC utility program described in this section.

Notes:

1. See the OS publication Programmer's Guide to Debugging.
2. See the TCAM Program Logic Manual.
3. See TCAM I/O Error-recording Facility in this chapter.
4. See Using TCAM's Message Logging Facility elsewhere in this publication.

Figure 36. Coding Requirements for Using TCAM Debugging Aids.

On-line Test Function

The on-line test (OLT) function is an optional TCAM facility; its implementation is described in detail in the *TOTE/Configurator User's Manual.* OLT permits either a system console operator or a remote control station user to test transmission control units and remote stations. The OLT function is used to:

- Diagnose hardware errors
- Verify repairs
- Verify engineering changes
- Check devices periodically
- Check new stations brought on-line

The TCAM OLT function is implemented in three parts:

- the telecommunications on-line test executive (TOTE)

TOTE acts as an interface between TCAM and the OLTs for scheduling and controlling the execution of OLTs. TOTE also prompts the user when he requests help, when he makes an invalid request, or when a test needs more data.

- a configurator

The configurator collects data about the stations and control units from TCAM and OS. When adequate data is not available the configurator prompts the on-line test user for the information.

- On-line tests (OLTs)

The OLTs run under the supervision of TOTE. They reside in either SYS1.LINKLIB or a private library. The results are sent to a station specified by the test requester. The device tests are written primarily by taking the off-line diagnostics and converting them to run on-line under TCAM/TOTE. On-line tests affect application performance to the extent that test transmissions require line time, tests require CPU time, and OLT modules require main storage and DASD space.

Advantages of TOTE

In order to properly assess the resources required to support TOTE, it is necessary to put TOTE in the proper perspective. TOTE is advantageous because it provides:

- Dynamic remote test request
- Remote test control
- BSC support
- Operation in a dedicated TP system

Devices Supported

On-line tests are provided for the IBM devices listed in the section *Machine and Device Requirements* of the chapter titled *System Preparation.* In addition, on-line tests for the following adapters are provided:

2701 adapters and features:
IBMI, IBMII, IBMIII, SDAII, TTYI, TTYII, and WTC-TTY

2702 adapters and features:
IBMI, IBMII, TTYI, TTYII, and WTC-TTY

2703 adapters and features:
IBMI, IBMII, TTYI, TTYII, BSC, and WTC-TTY

TOTE Facilities

Two levels of tests are supported by TCAM/TOTE. These levels are referred to by the name of the message used to request them. These are defined as:

RFT (request for test). This refers to a test whose function is to determine if a device works or does not work and may be used as a tool in the diagnostic activity. These are similar to those provided by OS/BTAM.

TRM (terminal request message). This refers to a test whose function is to perform problem definition. These tests provide functional testing and are, for the most part, written by converting existing off-line diagnostics to run on-line under TCAM/TOTE.

With respect to the application's use of the TP sub-system, on-line tests under TOTE may be run in two modes, concurrent and nonconcurrent. Concurrent mode means that TOTE can execute an on-line test for one station on a multipoint line while the application continues to use the remaining stations. Nonconcurrent mode means an entire line must be dedicated to the on-line test function. BSC tests normally cannot be run in concurrent mode.

Synchronous testing, while not an explicit feature of on-line tests under TCAM, is a by-product of the design (maintenance support is provided for this facility). This is defined as the execution of the same or different OLTs on different devices during the same time interval. Special coding considerations are not required. Enough main storage must be allocated to support the maximum number of tests to be run asynchronously during the same time interval.

It is possible for test request messages to be entered at the time an on-line test is being executed (or the maximum number of asynchronous tests are being executed). In this case, the test request message is queued and the test requester is notified that it has been queued.

Other TOTE facilities that support on-line test execution are:

- Repeating a given test a specified number of times (testing loop).
- Looping on the set of instructions that detect the first error (error loop).
- Printing options such as notification of testing progress, detailed error prints, and an alternate printer.
- Executing test routines that require manual intervention during their execution.
- Entering test requests from the system console as well as from a remote station.

System Requirements

This section describes main storage requirements, TCAM MCP facilities that must be specified to support on-line testing, OS/SYSGEN options that must be specified, and JCL requirements for TOTE/OLTs.

Main-storage Requirements: TOTE plus the OLT reside in the MCP partition or region of TCAM. The basic main storage requirement is 10K bytes. This allocates enough space for TOTE (6K) plus one OLT section (4K). Test sections for BSC and display devices may require 6K to 8K bytes.

If asynchronous testing is desired, more storage is required. The following formula should be used to calculate the specific number of bytes:

$$n(6 + MTS)$$

where: n=number of tests to be run.
MTS=maximum on-line test size.
The local FE Branch Office provides assistance in determining the MTS value for specific BSC or display devices.

The main storage occupied by the OLTs may be shared with other TCAM functions such as checkpoint/restart. This storage is not available to the application.

TOTE Requirements: The OLTEST= operand of the INTRO macro instruction must either be omitted or must specify either zero or a positive integer greater than 9. If the operand is omitted, 10K bytes of storage is reserved for on-line tests. If OLTEST=0 is

coded, *no* storage is reserved. If OLTEST=10 is coded, 10K bytes are reserved; if OLTEST=21 is coded, 21K bytes are reserved, etc. At least 10K bytes of main storage must be reserved if on-line test is to be incorporated into the TCAM system.

The STARTMH routine determines if incoming messages are OLT requests. When STARTMH recognizes a message as an OLT request, and TOTE is not active because the OLTEST= operand of the INTRO macro instruction specifies zero, TCAM sets bit 12 in the message error record to indicate that TOTE is not active. This bit is also set by TCAM when the amount of storage specified by the OLTEST= operand is currently being used by TOTE, and when the OLT request does not fit into a single buffer (see next section).

Coding Requirements: Each on-line test request must fit within a single buffer; furthermore, an on-line test message identifier (either SOH% or 99999) must fit within the first buffer unit. These buffer design condiderations must be taken into account when the user specifies buffer and buffer unit sizes (see the KEYLEN= and LNUNITS= operands of the INTRO macro instruction and the BUFSIZE= operand of the line group DCB macro instruction).

In designing his MH, the user should determine if bit 12 is on when an OLT request is made; if this bit is on, he should send an error message to the requesting station (see the descriptions of the MSGGEN and ERRORMSG macro instructions). It is up to the station operator who receives the error message to determine that:

1. TOTE is in the system (that is, the on-line test function was specified properly at execution time), and
2. the test request fits into a single buffer (that is, the system programmer specified TCAM buffers large enough to hold an OLT request).

If these two requirements are met, then the station operator should retry his request later (his request was not honored because the amount of main storage specified for on-line test was being used to accomplish other on-line tests that were being made). If these two requirements are not met, an on-line test cannot be performed.

NOTE 1: When a group of terminals in a TCAM environment are defined by the UTERM=YES operand of the TERMINAL macro instruction, there is only one symbolic name associated with all the terminals on that line. This symbolic name can be used by TOTE in its scheduling and control functions provided that all terminals on the line are the same type.

NOTE 2: When STARTMH recognizes a TOTE request message, the line associated with the station entering the request is marked stopped. If an operator command is issued to stop the line a response is received stating that the line is already stopped. If the request is for another line, the original line is restarted, and the requested line is stopped.

OS/SYSGEN Requirements: In order to support TOTE, the SUPRVSOR macro instruction must specify OPTIONS=ONLNTEST at system generation time.

At least two buffers must be specified in the WTOBFRS parameter of the SCHEDULR macro instruction when on-line tests are included in the generated system.

JCL Requirements for TOTE/OLTs: The following DD statements must be included in the TCAM JCL that defines data sets when the on-line test function is to be included in the TCAM system:

- A DD card for output of diagnostic messages from TOTE/OLT to SYSOUT:

```
//DIAGMSG      DD SYSOUT=A
```

NOTE: To obtain direct output of TOTE diagnostic messages when APSYSOUT is specified, substitute UNIT=xxx for SYSOUT=A. *xxx* is the 3-digit hexadecimal address of a printer that is not assigned to an output writer.

- A DD card for the OLT unit test module library:

```
//JOBLIB        DD DSN=SYS1.OLTLIB,DISP=SHR
```

NOTE: This example assumes that the OLT unit test modules have been replaced in a private library named SYS1.OLTLIB, and SYS1.OLTLIB has been cataloged.

- A DD card for the configuration data library:

```
//DCHBDD        DD DSN=SYS1.OLTLIB(DCHB),DISP=OLD
```

NOTE: This example assumes that the configuration data is to be placed in SYS1.OLTLIB as a member named DCHB. It also assumes that SYS1.OLTLIB has been cataloged.

This chapter provides information needed in setting up a teleprocessing system to be run under TCAM. It indicates the machine and device requirements of a TCAM system, touches upon system generation requirements peculiar to TCAM, and describes the IEDQXA utility program provided by TCAM for preformatting message queues data sets on disk. The *System Generation* publication provides information for generating an IBM System/360 Operating System, including machine configuration and data processing requirements.

Machine and Device Requirements

TCAM operates under the operating system MFT-II and MVT environments on any System/360 Model 40 or above (that is, a CPU having at least 128K of main storage). The only additions to the minimum requirements of the System/360 Operating System are:

- All telecommunications terminals, except the IBM 2260-2848 Local, must be attached to either an IBM 2701 Data Adapter Unit Model 1, an IBM 2702 Transmission Control Model 1, an IBM 2703 Transmission Control Model 1, or an IBM 7770 Model 3 Audio Response Unit; they cannot be attached directly to a channel.
- All IBM 2701, 2702, 2703, or 7770 control units that operate under TCAM must be attached to the System/360 through the multiplexer channel.

NOTE: A switch on the CE panel on the 2702 can be used to place a given line in CE mode for equipment checking. Care must be taken to ensure that no lines are in CE mode when TCAM is used since no ending status will be returned to an SIO command that is issued by the system.

- No device may be operated in burst mode on the multiplexer channel concurrently with the operation of TCAM, except when the TCAM operation involves only the 2260 Display Complex (Local).

The following additional features may be required:

- The system ATTACH macro instruction must be specified for an MFT system;
- The line correction feature on IBM 1050 Data Communication System terminals if automatic retry is desired when a transmission error occurs.

Control Units and Terminal Types Supported

TCAM supports any combination of the IBM 7770 Audio Response Unit and the IBM 2701, 2702, or 2703 transmission control units on the same multiplexer channel. Up to eight control units can be attached directly to the multiplexer channel. TCAM also supports the IBM 2848 Display Control attached directly either to the multiplexer or a selector channel. Figure 37 below illustrates the device configurations supported by TCAM.

Multiprocessing System

TCAM supports the multiprocessing (IBM Model 65 MP) system with the configuration-control feature; this system is formed from two Model 65s operating as a single large-scale system under one control program. Since the Model 65 MP permits simultaneous execution of two tasks in the system, the TCAM MCP can execute simultaneously with a TCAM application program.

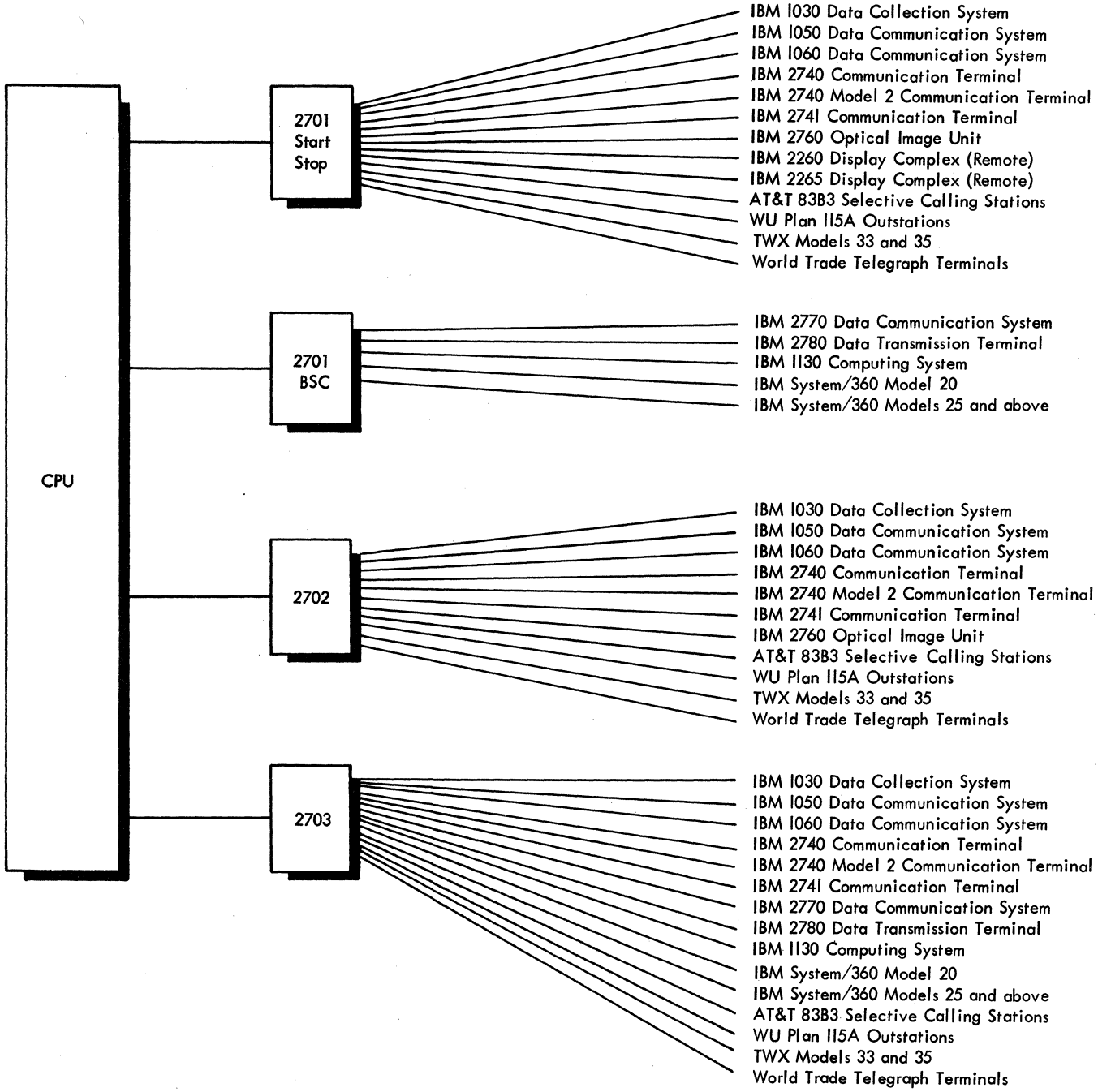


Figure 37. Device Configurations Supported by TCAM (Part 1 of 3)

Station Type		Channel Type		TCU			Audio Response Unit	Line Type		Notes
		Multiplexer	Selector	IBM 2701 Data Adapter Unit	IBM 2702 Transmission Control	IBM 2703 Transmission Control	IBM 7770 Model 3	Switched	Nonswitched	
IBM 1030 Data Collection System	Auto Poll	X			X	X			X	The IBM Digital Time Out feature cannot be attached through an IBM 2701 TCU.
		X		X	X	X			X	
IBM 1050 Data Communication System	Auto Poll	X			X	X			X	
		X		X	X	X		X	X	
IBM 1060 Data Communication System	Auto Poll	X			X	X			X	
		X		X	X	X			X	
IBM 2260-2848 Display Complex (Remote)		X		X					X	
IBM 2260-2848 Display Complex (Local)		X	X							
IBM 2265-2845 Display Complex (Remote)		X		X					X	
IBM 2740 Model 1 Communication Terminal	Auto Poll	X			X	X			X	Two Types: 2740 with station control 2740 with station control and record checking
		X		X	X	X			X	Four Types: 2740 basic 2740 with station control 2740 with record checking 2740 with station control and record checking
		X		X	X	X		X		Four Types, all with dial: 2740 2740 with transmit control 2740 with record checking 2740 with transmit control and record checking
IBM 2740 Model 2 Communication Terminal	Auto Poll	X			X	X			X	Four Types: 2740 2740 with record checking 2740 with buffer receive 2740 without buffer receive (requires line slowdown feature)
		X		X	X	X			X	Four Types: 2740 2740 with record checking 2740 with buffer receive 2740 without buffer receive
IBM 2741 Communication Terminal		X		X	X	X		X	X	The attention feature is not supported, and the break feature is supported only if the CPU is sending and the terminal has not entered data when the break is issued.

Figure 37. Device Configurations Supported by TCAM (Part 2 of 3)

Station Type		Channel Type: Multiplexer	Channel Type: Selector	TCU: IBM 2701 Data Adapter Unit	TCU: IBM 2702 Transmission Control	TCU: IBM 2703 Transmission Control	Audio Response Unit: IBM 7770 Model 3	Line Type: Switched	Line Type: Nonswitched	Notes
IBM 2760 Optical Image Unit								X	X	Attached to a 2740 Model 1 with record checking
IBM 2770 Data Communication System		X		X		X		X	X	BSC transmission using either ASCII or EBCDIC code
IBM 2780 Data Transmission Terminal		X		X		X		X	X	BSC transmission using ASCII, EBCDIC, or 6-bit code
IBM 1130 Computing System		X		X		X		X	X	BSC transmission
IBM System/360 Model 20		X		X		X		X	X	BSC transmission using either ASCII or EBCDIC code
IBM System/360 Models 25 and above		X		X		X		X	X	BSC transmission and point-to-point lines only
AT&T 83B3 Selective Calling Stations		X		X	X	X			X	
Western Union Plan 115A Outstations		X		X	X	X			X	
TWX Models 33 and 35		X		X	X	X		X		Teletype terminals, dial service (8-level code)
World Trade Telegraph Terminals		X		X	X	X			X	Control unit must incorporate a WTTA
Audio terminals		X					X	X		Example: IBM 2721 Portable Audio Terminal

Figure 37. Device Configurations Supported by TCAM (Part 3 of 3)

System Generation Considerations

TCAM facilities can be incorporated into an operating system by performing an operating system generation. This procedure is explained in the OS publication *System Generation.*

Using system generation macro instructions, the user specifies the line configuration and device requirements of the telecommunications system being supported, and any optional features required. TCAM is specified as an option in the ACSMETH= operand of the DATAMGT macro instruction.

The GENERATE macro instruction is modified for TCAM:

GENERATE: 1) A DD card will be punched for the SYS1.CQ548 component library before group three macros are called so that the attention routine to handle 2260/2848 interrupts can be link edited into the nucleus. 2) A DD card will be punched for the SYS1.CQ548 component library before the group five macros are called, so that modules can be selected from it by SGIEC5TP. 3) SGIEC3TP and SGIEC5TP will be called if QTAM, BTAM, or TCAM is specified.

IODEVICE and *IOCONTRL:* TCAM will support the 7770 Model 3 and 2845/2701.

There are four types of System Generation; three of these affect the DATAMGT macro and therefore TCAM.

1. Complete Operating System Generation: ACSMETH=(TCAM) must be specified and the TELCMLIB macro specified.
2. Nucleus Generation: ACSMETH= (TCAM) must be respecified so current SVCs are retained when the nucleus is re-link edited. If TCAM is not respecified, it will not be in the system generated by the nucleus generation. The TELCMLIB macro need not be respecified, but must have been specified in the Complete Operating System Generation.
3. Processor/Library Generation: No affect on TCAM.
4. I/O Device Generation: ACSMETH=(TCAM) must be respecified so current SVCs are retained. The TELCMLIB macro need not be respecified but must have been specified in the Complete Operating System Generation.

MFT users must allow enough SYSQUE space at SYSGEN time for attached subtasks – 180 bytes per ATTACH. The four attached subtasks are Checkpoint, Operator Control, Comwrite, and On-Line Test. Checkpoint, Comwrite and On-Line Test are optional. TCAM will ABEND if enough space is not allocated.

Preformatting DASD Message Queues Data Sets

Since TCAM expects the disk message queues (both reusable and nonreusable) to be totally preformatted, the IEDQXA routine should be used to perform this task prior to initially running any TCAM job (TCAM automatically formats the disk message queues for either a warm or a continuation restart).

At SYSGEN time, this routine is moved from SYS1.CQ548 to SYS1.LINKLIB along with other TCAM non-resident modules. As a utility program, it is executed in a separate step from the step executing TCAM.

```
//jobname     JOB
//stepname    EXEC PGM=IEDQXA
//SYSPRINT    DD   SYSOUT=A
//IEDQDATA    DD   DSNAME=anyname,DISP=(,CATLG),          *
//                   SPACE=(CYL,(n,n),,CONTIG),          *
//                   UNIT=(23xx,y),                      *
//                   VOLUME=SER=(aaaaaa,bbbbbb,...),     *
//                   DCB=(,KEYLEN=mm)
```

Figure 38. Sample JCL for IEDQXA Utility

The variables in the figure are defined as follows:

anyname

User selects any name for the data set.

n

Since the number of cylinders must be the same for all extents, both primary and secondary allocations must be identical. Allocation must be by cylinders.

xx

Any one disk message queue must have all extents on one type of disk, either all on 2311 or on 2314 type disks.

y

The total number of volume serial numbers listed in VOLUME parameter. Maximum is 16.

aaaaaa,bbbbbb,...

There is one extent per volume. List each volume serial number of each volume to contain one extent of the data set. *aaaaaa* is the first extent, *bbbbbb* is the second, and so on. A maximum of 16 volumes may be identified.

mm

Each record formatted contains a *key* and a *data* field. *mm* is the size of each key portion of the record. Contrary to traditional usage of these fields, TCAM sets up a short *data* field for internal control information and places the actual buffer data in the *key* field. The *data* field size is an internally fixed constant (6 bytes), and the size of the *key* field (i.e., *mm*) must be identical with the buffer unit size, as specified by the KEYLEN= operand of the INTRO macro. Although the buffer unit in main storage contains an internally generated 12-byte prefix, only the first six of these bytes are used to define the data field on disk. (Guidelines for determining a suitable buffer unit size are contained in the chapter *Defining Buffers*.)

There is no difference in the creation of reusable or nonreusable disk message queues. The data set created by this routine may be used by TCAM as either one.

In addition to the desired data set, this routine lists a statement on the SYSPRINT data set after the successful formatting of each extent. An accumulative record total is listed for each extent. This data set contains any error messages or an indication of successful completion.

Appendix A. TCAM Macro Formats

A format illustration accompanies each macro instruction in this publication. The illustrations indicate which operands must be coded exactly as shown, which are required, which are variable, etc. The conventions stated to describe the operands are as follows:

1. Keyword operands are described by a three-part structure that consists of the (uppercase) keyword operand, followed by an equal sign (both of which must be coded), followed by a lowercase variable or an uppercase fixed value to be specified by the user. Examples: KEYWORD=value, METHOD=NORMAL
2. Positional operands are described by a lowercase name, which is merely a convenient reference to the operand and is never coded by the programmer, or by an uppercase operand that is used exactly as shown. The programmer replaces the lowercase operand by an allowable expression as defined in the macro description. Examples: *qtype, destname, mask,* MESSAGE.
3. Uppercase letters and punctuation marks (except as described in these conventions) represent information that must be coded exactly as shown.
4. Lowercase letters and terms represent information that must be supplied by the programmer. Restrictions (such as the maximum value that may be specified) are stated in the description of the operand under the macro description.
5. An ellipsis (a comma followed by three periods) indicates that a variable number of items may be included.
6. {A / B} Options contained within braces represent alternatives, one of which is chosen by the user when he codes the operand in which the braces appear.
7. [A / B] Information contained within brackets represents an option that can be included or omitted, depending on the requirements of the program. If more than one alternative is included within a single set of brackets, either of the alternatives may be chosen, or the operand may be omitted (i.e., none of the alternatives are chosen). Operands that are not enclosed within brackets are required.
8. [A / B / C] Underlined elements represent an assumed value in the event a parameter is omitted.

Conventions Used

In describing and illustrating the coding of macro instructions, the following conventions are used.

Register notation: Unless otherwise specified, a register (2 through 11) may be used. The number of the register must be enclosed in parentheses.

Error Returns: Error return codes are returned in the right-hand byte of register 15.

Commas in Operand Field: Sometimes two optional keyword operands are listed, such as [A][,B]. The comma is to be omitted if A is omitted. The comma must not be omitted for positional operands that are specified in another way.

Blanks in Operand Field: No blanks are allowed within the operand field.

Appendix B: Message Error Record

A five-byte *message error record* is assigned by TCAM to each message for the duration of its processing by the incoming or outgoing group of a Message Handler; this message error record may be checked by macros coded in the inmessage or outmessage subgroup of that group. Each of the 40 bits of the message error record (except reserved bits) indicates the presence (when 1) or the absence (when 0) of a specific error condition that has affected or may affect successful processing or transmission of the message. Some of the errors that may be recorded in the message error record are transmission and equipment errors (e.g., lost data, busout check), some are due to mistakes in entering a message (e.g., wrong sequence number, invalid origin code), and some are due to a shortage of system resources (e.g., insufficient number of buffers, insufficient space in a main-storage-only message queues data set).

The TCAM user may code one or several error-handling macros in his Message Handler; among these are CANCELMG, ERRORMSG, MSGGEN, REDIRECT, and HOLD. CANCELMG may be coded in the inmessage subgroup only, while the others may be coded in either the inmessage or the outmessage subgroup. These macros each have an optional five-byte error-mask operand, which may be used to test the message error record, so that the error-handling macro for which the mask is coded is executed only if the errors specified in the mask have occurred. When error-handling macros are coded in an inmessage subgroup, they test the message error record after the message is received from a station or application program; in the outmessage subgroup they test the message error record after the message is sent to a station or application program.

The last byte of the message error record consists of the sense byte for the I/O device (in this case, the transmission control unit being used). When the unit check bit is turned on in the CSW during an I/O operation, a sense command is issued by TCAM, and the appropriate bits in the sense byte are turned on. The CSW and the sense command are described in the *Principles of Operation.* A detailed discussion of the meaning of each bit in the sense byte may be found in the component description SRL for the transmission control unit being used.

The meaning of each bit in the message error record is shown below. Bit 0 is the left-most bit and Bit 39 the right-most bit in each error record.

Bit — *Meaning*

0 — Header error

The scan pointer has reached the end of the last segment in the message, but the end of the inheader or outheader subgroup has not been reached.

1 — Invalid origin code

The ORIGIN macro found that the origin field in the incoming header contained a code that:

a. did not correspond to the name of a station that was connected to the computer over a nonswitched line, or
b. did not correspond to any station name in any group (applicable only to stations on switched lined).

2 — reserved

3 — Sequence number high or not a valid number

The SEQUENCE macro found a message sequence number that is not a valid decimal integer or is higher than the expected number for the next message originating from the station. When this error is detected, the expected sequence number is not changed. If the message is not canceled by the user, the same sequence number may appear in more than one message.

Bit	*Meaning*
4	Sequence number low The SEQUENCE macro found a message sequence number lower than the expected number for the next message originating from the station. The user may inadvertently use the same message number in more than one message. This bit can be used to detect such an error, thus allowing the user to re-send the corrected message.
5	reserved
6	Insufficient buffers The TCAM buffer assignment routine was unable to provide sufficient buffers for the incoming message. Infrequent occurrences of this condition may be corrected by requesting the originating station to re-send the message. Frequent occurrences of this condition suggest that TCAM be redefined with a larger number of buffers.
7	Cutoff error The CUTOFF macro found a buffer filled with identical characters or a message whose length exceeded the maximum allowable length.
8	MSMIN passed The percentage of the number of units specified by the MSUNITS= operand of the INTRO macro that are currently enqueued in the main-storage message queues data set has fallen to or below the number specified by the MSMIN= operand of INTRO; used to indicate impending failure of the data set.
9	MSMAX passed The percentage of the number of units specified by the MSUNITS= operand of the INTRO macro that are currently enqueued in the main-storage message queues data set has risen to or above the number specified by the MSMAX= operand of INTRO; used to indicate impending fullness of the data set.
10	reserved
11	reserved
12	reserved
13	TOTE not in system A request for on-line test has been detected by a STARTMH macro, but TOTE is not included in the OLT= operand of the INTRO macro instruction.
14	BSC abort An abort sequence was received from a BSC station.
15	Invalid destination code A destination specified in the FORWARD macro is invalid because it does not have a matching entry in the terminal table.
16	Incoming message lost An incoming message has been lost due to lack of space in a main-storage-only message queues data set.

Bit	*Meaning*
17	Invalid station identification An identification sequence sent from a station is invalid.
18	Station inoperative The destination station for this message is in intercept mode, and messages are not currently being sent to it.
19	reserved
20	User error This bit may be set by the user to indicate a logical error condition of his choosing. The bit is set by means of a TERRSET macro issued in a Message Handler.
21	Format error Message from BSC station is in wrong format for BSC (for instance, text does not start with the required STX character).
22	reserved
23	Unit exception The unit exception bit is on in the CSW, indicating the presence of a condition that does not usually occur during an I/O operation.
24	Error during invitation or selection An error occurred during invitation or selection (before text transfer).
25	Error during text transfer An error occurred during transfer of data.
26	Error during connection or disconnection An error occurred before invitation or selection, or while attempting to disconnect.
27	reserved
28	reserved
29	Error in control unit A busout, equipment check, overrun, or similar error, recognized by the control unit as an error in the control unit, has occurred.
30	Error in channel A channel control check, interface control check, channel data check, or command reject has occurred.
31	Undefined error An error has occurred that cannot be classified by TCAM.

SENSE BYTE

Bit	*Meaning*
32	Command reject A command or a series of commands is received that the device is not designed to execute or cannot execute because of its present state.
33	Intervention required Some sort of intervention is required, or the device is in the not-ready state, or in test mode, or not on the control unit.
34	Busout check An invalid parity character is received by the device or control unit.
35	Equipment check The device has malfunctioned.
36	Data check An error has occurred associated with the recording medium.
37	Overrun The channel failed to respond on time to a request for service from a device, or a device received a new command too late during command chaining.
38	Lost data
39	Time-out exceeded More than the maximum allowable time elapsed between polling or addressing a station and reception of a response from it.

Appendix C: How To Make Transient Checkpoint and Operator Control Modules Resident

Certain TCAM modules connected with the TCAM checkpoint and operator control facilities are normally transient, but may be made resident if the user so desires. By making frequently used modules resident, the user increases the performance of his system, at the expense of additional main-storage space.

The TCAM checkpoint and operator control modules that may be made resident are located in SYS1.LINKLIB. In order to make these modules resident, the user must first specify at system-generation time the Reenterable Load Module Made Resident option. This is done by specifying the OPTIONS=COMM and RESIDNT=RENTCODE operands of the SUPRVSOR system generation macro; details are given in the *System Generation* publication.

Before initial program loading (IPL), the user makes a list of the load modules he wishes to make resident and places it in SYS1.PARMLIB by means of the IEBUPDTE utility program. For more information on placing such a list in SYS1.PARMLIB, see the *System Programmer's Guide.*

At IPL time, in response to the console message SPECIFY SYSTEM PARAMETERS, the operator provides the unique identification for the list, and the routines pointed to by the list are loaded into main storage. More information on replying to this message is contained in the *Messages and Codes* publication.

The following checklists show,

a. in order of decreasing frequency of use, the ordinarily transient modules associated with TCAM checkpoint routines that may be made resident as described above;
b. ordinarily transient modules associated with operator commands that may be made resident. Some operator commands have more than one such module associated with them.

Example:
An MVT user wishes to make the routines for the ENTERING, ACTVBOTH, ACTVATED, NOENTRNG, and NOTRAFIC operator commands resident. At system generation time the user must code the RESIDNT= operand of the SUPRVSOR macro RESIDNT=RENTCODE, and the OPTIONS= operand of the same macro OPTIONS=COMM.

The user might create a list named IEAIGGOC to contain the modules for these messages. He would get the modules from the checklist of modules and their sizes. Sometime before IPL, he might use the IEBUPDTE utility program as shown in the example of using the IEBUPDTE utility to place the list in SYS1.PARMLIB.

```
//ADDLIST       JOB       1865,R.E.LEE
//STEP          EXEC      PGM=IEBUPDTE,PARM=NEW
//SYSPRINT      DD        SYSOUT=A
//SYSUT2        DD        DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN         DD        DATA
./              ADD       NAME=IEAIGGOC,LIST=ALL
./              NUMBER    NEW1=01,INCR=02
 SYS1.LINKLIB             IEDQCO,IEDQCL
./              ENDUP
/*
```

Figure 39. Sample of Using the IEBUPDTE Utility (prior to IPL) for Placing a List in SYS1.PARMLIB

Note that the statement beginning SYS1.LINKLIB begins in column 2.

At IPL time, in response to the console message

IEA101A SPECIFY SYSTEM PARAMETERS

the operator might reply

REPLY id, 'RAM=00,OC'

where 00 are the last two characters in the name of the standard list of names of reenterable load modules, and OC are the last two characters in the name of the IEAIGGOC list.

Checkpoint Modules and Their Sizes

Module	*Description*	*Decimal Size Approximate*
IEDQNG	Incident Record for CHECKPT	250 bytes
IEDQNH	Incident Record for TCHNG	240 bytes
IEDQNJ	Checkpoint/Operator Control	240 bytes
IEDQNK	Environment Checkpoint	800 bytes
IEDQNM	CKREQ	390 bytes
IEDQNO	Checkpoint Queue Manager	240 bytes
IEDQNP	Checkpoint I/O	640 bytes
IEDQNQ	Checkpoint Notification/ Disposition	730 bytes
IEDQNR	No Main Storage	260 bytes
IEDQNS	Incident Overflow	160 bytes

Modules Associated with Operator Commands

Message	*Modules Used*	*Decimal Size Approximate*
ACTVATED	IEDQCL	1110 bytes
ACTVBOTH	IEDQCO	1510 bytes
AUTOSTOP	IEDQCW	890 bytes
AUTOSTRT	IEDQCW	890 bytes
CPRIOPCL	IEDQCN	530 bytes
DATOPFLD	IEDQCF	2270 bytes
DEBUG	IEDQC6	1370 bytes
DPRIOPCL	IEDQCM	550 bytes
DSECOPCL	IEDQCM	550 bytes
ENTERING	IEDQCO	1510 bytes
ERRECORD	IEDQCX	803 bytes
GOTRACE	IEDQCP	870 bytes
INACTVTD	IEDQCL	1110 bytes
INTERVAL	IEDQCZ	940 bytes
INTRCEPT	IEDQCK	510 bytes
LNSTATUS	IEDQCI	1070 bytes
NOENTRNG	IEDQCO	1510 bytes
NOTRACE	IEDQCP	870 bytes
NOTRAFIC	IEDQCO	1510 bytes
OPTFIELD	IEDQCF	2270 bytes
POLLDLAY	IEDQCZ	940 bytes
QSTATUS	IEDQCJ	680 bytes
RESMXMIT	IEDQCQ	1170 bytes
RLNSTATN	IEDQCG	510 bytes
STATDISP	IEDQC3	60 bytes
STARTLINE	IEDQCU	1720 bytes
STOPLINE	IEDQCV	1230 bytes
STSTATUS	IEDQCH	700 bytes
SUSPXMIT	IEDQCQ	1170 bytes
SYSCLOSE	IEDQCO IEDQCV	1230 bytes
SYSINTVL	IEDQCZ	940 bytes

Two sets of charts are included in this appendix. Figure 40 comprises four foldout charts that include character sets and hexadecimal code for the extended binary coded decimal interchange code (EBCDIC), line codes for BSC devices (USASCII and 6-bit Transcode hexadecimal representations that correspond to EBCDIC), and line codes for start-stop devices (hexadecimal representations that correspond to EBCDIC). Figures 41 through 55 compose the second set of code charts; these figures illustrate in collating sequence (from hexadecimal 00 to hexadecimal FF) the valid hexadecimal representations of graphic and control characters for each device.

The first set of charts (Figure 40) is based on the collating sequence of the EBCDIC code that is used internally by the Operating System/360 Central Processing Unit (see column 3). Line code for BSC devices may be in either EBCDIC, USASCII, or 6-bit Transcode (columns 1 through 9). Columns 10 through 45 represent the character and code sets for start-stop devices that correspond to the internal EBCDIC code listed in column 3.

Arrangement of Charts

There are three columns associated with each entry in Figure 40. For instance, columns 1, 2, and 3 are associated with the EBCDIC entry, and columns 10, 11, and 12 are associated with the IBM 1030 entry. The unnumbered columns on the left and right ends of Figure 40 are reference numbers to designate rows. These numbers can be used in conjunction with the column numbers to designate a particular entry on the chart; for instance, location 21/17, the intersection of row 21 and column 17, contains the control character CR (carriage return) for the IBM 1060 Data Communication System. For ease of reference, column 46 repeats the EBCDIC code that appears in column 3.

The arrangement of the charts in Figures 41 through 55 is based on the collating sequence of the hexadecimal representations of the line code for the various devices.

Thus, columns 1 through 13 in Figure 40 (in conjunction with the columns that correspond to the device that originally entered the message) may be used for decoding messages in a dump when those messages have already been translated by the appropriate translation table; if a message was entered by a BSC device whose line code is EBCDIC, then columns 1 through 3 may be used for line code translation *and* internal System/360 translation. Figures 41 through 55 may be used to decode messages in a dump when the message appears in line code (incoming messages are in line code when they have not yet been translated to EBCDIC, and outgoing messages are in line code when they have been translated from EBCDIC).

Conventions Used In Code Charts

In the code columns for the various devices in Figure 40, some hexadecimal representations appear in parentheses, some in brackets, and others in neither. Where parentheses are used, only outgoing translation is performed by the translation table that corresponds to the device type for that column. For example, the alphabetical letter W in internal OS/360 code (EBCDIC) is represented by the bit pattern that corresponds to a hexadecimal E6 (see locations 230/1 and 230/3). If hexadecimal E6 is to be transmitted to an IBM 1060 Data Communication System, it must first be translated to the appropriate line code. The character is directed to the appropriate translation table where it is converted to hexadecimal 2C (see locations 230/16 and 239/18), which is the hexadecimal representation of the appropriate line code for the character W to be transmitted *to* an IBM 1060. Where the hexadecimal representation is enclosed in brackets (for instance, location 127/30), only incoming translation is performed; thus, hexadecimal 8E is translated to EBCDIC 7F when an IBM 2741 Communication Terminal using BCD code enters line code for the character double quote (see locations 127/3, 127/28, and 127/30). If there are neither parentheses nor brackets, both incoming and outgoing translation is provided for that device.

Various code set options are indicated in the graphic columns in Figure 40 and 41 through 55. Where S, H, A, and C appear as subscripts to a character, S indicates that the TCAM-provided translation table supports use of the standard code set for that device; H, A, and C indicate TCAM support of optional code sets for that device. For instance, at location 2/12, hexadecimal 16 is the outgoing line code to an IBM 1030

Data Collection System; the graphic characters at location 2/10 indicate that a pound sign (#) is printed by the IBM 1033 Printer if the printer uses the standard character set, and an equal sign (=) if it uses the H option. See the component description SRL of the device for a description of the character sets that may be used (TCAM supplies translation tables for AT&T TWX terminals that use the standard option, and for AT&T 83B3 and Western Union 115A terminals that use either A or C options).

Because each unique bit pattern for a terminal character can be represented only once in an *incoming* translation table, the character associated with the bit pattern can be translated to only one EBCDIC character. The converse is not true, however; any one transmission code bit pattern can be placed any number of times within an *outgoing* table. Therefore, any number of EBCDIC characters can be translated to the terminal character represented by that bit pattern.

Appearance of two bit patterns opposite a single character signifies that the character has both an uppercase (or figures shift) and a lowercase (or letters shift) bit pattern, and that both forms of the character are translated to the same EBCDIC character. (Exception: In the code column for TWX terminals, where two bit patterns appear, the left-hand one is the even-parity pattern, and the right-hand one is the non-parity pattern.)

Example:
The bit pattern of the NL character appears in location 21/15. Both the lowercase and uppercase bit patterns of this character are translated to the EBCDIC NL character when they appear in an incoming message. When an EBCDIC NL character appears in an outgoing message, TCAM translates it to the lowercase form of the NL character.

Where more than one EBCDIC character requires translation to the same character in a terminal character set, the terminal character appears an equivalent number of time in the column (for instance, locations 0/35, 6/35, 7/35, 23/35, and 50/35 all contain the LTRS character).

Where a character appears in both the graphics and the controls columns for a terminal type, its function depends on whether it is sent when the line is in control mode or text mode. Depending on the type of terminal and the mode, the character may perform a control function, print as a graphic, or both. For details, see the reference manuals for the various terminal types.

Non-equivalent Characters

Designing the system to accommodate terminal types having different character sets and control functions has resulted in several instances where dissimilar characters have been *equated* in translation tables. This accounts for the appearance in certain rows in Figure 40 of non-equivalent characters, for example, in rows 3, 38, and 50.

In other instances, the same or similar functions have different names among the various terminal types; for example, HT and Tab in row 5 are equivalent, as are DEL and Rubout in row 7. In a few instances, terminals using the same transmission code have different meanings assigned to the identical bit pattern; for example, bit pattern 79 in the transmission code has the meaning PF for an IBM 1050, and Subtract for an IBM 1060.

Substitutions

Where blank positions appear in the character columns of the charts, there is no equivalent internal EBCDIC character. Where these blanks appear, the SUB character is to be assumed (they were omitted to make the charts more readable). That is, in each translation table that handles incoming messages, each position representing an invalid transmission code bit pattern (that is, one not specified in the terminal's character set) is translated to the EBCDIC code 3F for the SUB character. In each translation table that handles outgoing messages, the transmission code bit pattern for a substitute graphic is contained in each of the following positions:

- Each position that represents an invalid EBCDIC bit pattern (a pattern to which no EBCDIC characters have been assigned),
- Each position that represents a bit pattern for a character having no equivalent in the destination terminal's character set.

For the IBM 1050, 2260, 2740, and 2741, this substitute character is a colon (:). For the IBM 1030 and 1060, and the AT&T TWX and 83B3, and the Western Union 115A, it is a slash (/).

General Notes

Standard abbreviations are used to represent the control characters. The full names of the characters are given in the section *Control Characters* below. For descriptions of these characters, see the reference manuals for the various terminals.

Where a circle character (Ⓑ, Ⓓ, etc.) appears in parentheses adjacent to a control character, it is an alternate name for the control character.

Most of the characters in the S and H character set options (1030) and in the A and C character set options (83B3, 115A) are identical. Where they differ between the options, the translation tables *favor* the S option and the A option, as illustrated in the charts. If messages from an H option 1030 are sent only to another H option 1030, the translation table may be used as is, and similarly, for the 83B3/115A, with respect to the C option. If messages from terminals with the H or C option are to be exchanged with other terminal types, the user should provide his own translation tables.

Control Characters

ACK	Positive Acknowledgment
Ⓑ	End-of-block (same as EOB)
BEL	Bell
BS	Backspace
BYP	Bypass
Ⓒ	End-of-transmission (same as (EOT)
CAN	Cancel
CC	Cursor control
CR	Carriage (carrier) return
CU1, CU2, CU3	Reserved for customer use
Ⓓ	Machine end-of-address (same as EOA)
DC1, DC2, DC4	Device link escape
DEL	Delete
DLE	Data link escape
DS	Digit select
EM	End of medium
ENQ	Enquiry
EOA	End-of-address
EOB	End-of-block
EOC	End of card
EOFC	End of first card
EOM	End-of-message
EOT	End-of-transmission
ETB	End-transmission-block
ETX	End-of-text
FF	Forms feed
FIGS	Figures shift
FS	Field separator
HT	Horizontal tabulate
IFS	Interchange file separator
IGS	Interchange group separator
IL	Idle
IRS	Interchange record separator
IUS	Interchange unit separator

LC	Lowercase shift
LF	Line feed
LF-CR	Line feed-carriage return
LTRS	Letters shift
MZ	Minus zero
Ⓝ	Negative response to polling, addressing, or LRC/VRC
NAK	Negative acknowledgment
NL	New line
NUL	Null
PF	Punch off
PN	Punch on
PRE	Prefix
PZ	Plus zero
RES	Restore
RM	Record mark
RS	Reader stop
Ⓢ	Start-of-address
SI	Shift in
SM	Set mode
SMI	Start manual input
SO	Shift out
SOH	Start-of-header
SMM	Start-manual-message
SOS	Start-of-significance
SP	Space
STX	Start-of-text
SUB	Substitute
SYN	Synchronous idle
Tab	Tabulate (horizontal)
TM	Tape mark
TpAuxOff	Tape auxiliary off
TpAuxOn	Tape auxiliary on
UC	Uppercase shift
VT	Vertical tabulate
WRU	'Who Are You?'
X-Off	Transmitter off
X-On	Transmitter on
Ⓨ	Positive response to polling, addressing, or LRC/VRC

S/360 Byte (hex)	Graphic	Control
00		NUL
01		SOH
02		STX
03		ETX
04		PF
05		HT
06		LC
07		DEL
08		
09		
0A		SMM
0B		VT
0C		FF
0D		CR
0E		SO
0F		SI
10		DLE
11		DC1
12		DC2
13		TM
14		RES
15		NL
16		BS
17		IL
18		CAN
19		EM
1A		CC
1B		CU1
1C		IFS
1D		IGS
1E		IRS
1F		IUS
20		DS
21		SOS
22		FS
23		
24		BYP
25		LF
26		ETB (EOB)
27		ESC (PRE)
28		
29		
2A		SM
2B		CU2
2C		
2D		ENQ
2E		ACK
2F		BEL
30		
31		
32		SYN
33		
34		PN
35		RS
36		UC
37		EOT
38		
39		
3A		
3B		CU3
3C		DC4
3D		NAK
3E		
3F		SUB

S/360 Byte (hex)	Graphic	Control
40		SP
41		
42		
43		
44		
45		
46		
47		
48		
49		
4A	¢	
4B	.	
4C	<	
4D	(	
4E	+	
4F	|	
50	&	
51		
52		
53		
54		
55		
56		
57		
58		
59		
5A	!	
5B	$	
5C	*	
5D	)	
5E	;	
5F	¬	
60	-	
61	/	
62		
63		
64		
65		
66		
67		
68		
69		
6A		EOM
6B	,	
6C	%	
6D	_	
6E	>	
6F	?	
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
7A	:	
7B	#	EOA
7C	@	
7D	'	
7E	=	
7F	"	

S/360 Byte (hex)	Graphic	Control
80		
81	a	
82	b	
83	c	
84	d	
85	e	
86	f	
87	g	
88	h	
89	i	
8A		
8B		
8C		
8D		
8E		
8F		
90		
91	j	
92	k	
93	l	
94	m	
95	n	
96	o	
97	p	
98	q	
99	r	
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2	s	
A3	t	
A4	u	
A5	v	
A6	w	
A7	x	
A8	y	
A9	z	
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		PZ
C1	A	
C2	B	
C3	C	
C4	D	
C5	E	
C6	F	
C7	G	
C8	H	
C9	I	
CA		
CB		
CC		
CD		
CE		
CF		
D0		MZ
D1	J	
D2	K	
D3	L	
D4	M	
D5	N	
D6	O	
D7	P	
D8	Q	
D9	R	
DA		
DB		
DC		
DD		
DE		
DF		
E0		RM
E1		
E2	S	
E3	T	
E4	U	
E5	V	
E6	W	
E7	X	
E8	Y	
E9	Z	
EA		
EB		
EC		
ED		
EE		
EF		
F0	0	
F1	1	
F2	2	
F3	3	
F4	4	
F5	5	
F6	6	
F7	7	
F8	8	
F9	9	
FA		
FB		
FC		
FD		
FE		
FF		

Figure 41. IBM S/360 Internal Code (EBCDIC)

S/360 Byte (hex)	Graphic	Control
00		NUL
01		SOH
02		STX
03		ETX
04		EOT
05		ENQ
06		ACK
07		BEL
08		BS
09		HT
0A		LF
0B		VT
0C		FF
0D		CR
0E		SO
0F		SI
10		DLE
11		DC1
12		DC2
13		DC3
14		DC4
15		NAK
16		SYN
17		ETB
18		CAN
19		EM
1A		SUB
1B		ESC
1C		FS
1D		GS
1E		RS
1F		US
20		SP
21	!	
22	"	
23	#	
24	$	
25	%	
26	&	
27	'	
28	(	
29	)	
2A	*	
2B	+	
2C	,	
2D	-	
2E	.	
2F	/	
30	0	
31	1	
32	2	
33	3	
34	4	
35	5	
36	6	
37	7	
38	8	
39	9	
3A	:	
3B	;	
3C	<	
3D	=	
3E	>	
3F	?	

S/360 Byte (hex)	Graphic	Control
40	@	
41	A	
42	B	
43	C	
44	D	
45	E	
46	F	
47	G	
48	H	
49	I	
4A	J	
4B	K	
4C	L	
4D	M	
4E	N	
4F	O	
50	P	
51	Q	
52	R	
53	S	
54	T	
55	U	
56	V	
57	W	
58	X	
59	Y	
5A	Z	
5B	[	
5C	\	
5D	]	
5E		
5F	_	
60	`	
61	a	
62	b	
63	c	
64	d	
65	e	
66	f	
67	g	
68	h	
69	i	
6A	j	
6B	k	
6C	l	
6D	m	
6E	n	
6F	o	
70	p	
71	q	
72	r	
73	s	
74	t	
75	u	
76	v	
77	w	
78	x	
79	y	
7A	z	
7B	{	
7C	|	
7D	}	
7E	~	
7F		DEL

S/360 Byte (hex)	Graphic	Control
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
CD		
CE		
CF		
D0		
D1		
D2		
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		
FB		
FC		
FD		
FE		
FF		

Figure 42. USASCII Code

Ref.	Internal S/360 Code (EBCDIC)			USASCII			6-Bit Transcode			IBM 1030			IBM 1050			IBM 1060		
	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)
	Graphic	Control		Graphic	Control		Graphic	Control		Graphic	Control		Graphic	Control		Graphic	Control	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0 1 2 3		NUL SOH STX ETX	00 01 02 03		NUL SOH STX ETX	00 01 02 03		SOH STX ETX	(3F) 00 0A 2E	' #S =H	Pad Ⓢ EOA(Ⓓ) EOB(Ⓑ)	(DF) (37) (16) (3D)	#	IL EOA(Ⓓ) EOB(Ⓑ)	(5E) (16) (3D)	#	IL EOA(Ⓓ) EOB(Ⓑ)	(5E) (16) (3D)
4 5 6 7		PF HT LC DEL	04 05 06 07		HT DEL	09 7F		HT DEL	2F 3F		HT Pad EOC DEL	(7A) (DF) 7F		PF Tab Dwnshft DEL	79 [F9] 7A [FA] 7C [FC] 7F [FF]		Subtr Tab IL DEL	(79) (7A) (5E) 7F
8 9 10 11		SMM VT	08 09 0A 0B		VT	0B												
12 13 14 15		FF CR SO SI	0C 0D 0E 0F		FF CR SO SI	0C 0D 0E 0F					LF-CR	(5B)		NL BYP RES	(5B) (38) (58)		CR	(5B)
16 17 18 19		DLE DC1 DC2 TM	10 11 12 13		DLE DC1 DC2 DC3	10 11 12 13		DLE	1F									
20 21 22 23		RES NL BS IL	14 15 16 17		LF BS NUL	(0A) 08 (00)		DEL	(3F)		LF-CR Pad	(5B) (DF)		RES NL BS IL	58 [D8] 5B [DB] 5D [DD] 5E [DE]	★	CR IL	(58) (5B) (5E)
24 25 26 27		CAN EM CC CU1	18 19 1A 1B		CAN EM	18 19		EM	3E									
28 29 30 31		IFS IGS IRS IUS	1C 1D 1E 1F		FS GS RS US	1C 1D 1E 1F		US	1D									
32 33 34 35		DS SOS FS	20 21 22 23															
36 37 38 39		BYP LF ETB(EOB) ESC(PRE)	24 25 26 27		LF ETB ESC	0A 17 1B		ETB ESC	0F 2A		LF EOB(Ⓑ)	(3B) 3D		BYP LF EOB PRE	38 [B8] 3B [BB] 3D [BD] 3E [BE]		LF EOB(Ⓑ)	(3B) 3D
40 41 42 43		SM CU2	28 29 2A 2B															
44 45 46 47		ENQ ACK BEL	2C 2D 2E 2F		ENQ ACK BEL	05 06 07		ENQ BEL	2D 0D									
48 49 50 51		SYN	30 31 32 33		SYN	16		SYN	3A		Pad	(DF)		IL	(5E)		IL	(5E)
52 53 54 55		PN RS UC EOT	34 35 36 37		EOT	04		EOT	1E		EOT(Ⓒ)	1F		PN RS Upshift EOT(Ⓒ)	19 [99] 1A [9A] 1C [9C] 1F		EOT(Ⓒ)	(1F)
56 57 58 59		CU3	38 39 3A 3B															
60 61 62 63		DC4 NAK SUB	3C 3D 3E 3F		DC4 NAK SUB	14 15 1A		NAK SUB	3D 0E	/		(23)	:		(88)	/		(23)
64 65 66 67		SP	40 41 42 43		SP	20		SP	1A		SP	01		SP	01 [81]		SP	01
68 69 70 71			44 45 46 47															

Ref.	IBM 2260 (Remote)/2265: 2260/2265			IBM 2260 (Remote)/2265: 1053			IBM 2740			2741: BCD			2741: EBCD			2741: Correspondence		
	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)	Character		Code (Hex)
	Graphic	Control		Graphic	Control		Graphic	Control		Graphic	Control		Graphic	Control		Graphic	Control	
	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
0 1 2 3		STX ETX	02 03		STX ETX	(02) (03)	/ #	IL Ⓢ EOA(Ⓓ) EOB(Ⓑ)	(5E) (37) (16) (3D)	/	IL Ⓢ EOA(Ⓓ) EOB(Ⓑ)	5E 37 16 [3D]						
4 5 6 7								HT Dwnshft DEL	7A [FA] 7C [FC] 7F [FF]		HT Dwnshft DEL	7A [FA] 7C [FC] [7F][FF]		HT LC DEL	7A [FA] 7C [FC] [7F] [FF]		Tab Dwnshft	7A [FA] 7C [FC]
8 9 10 11																		
12 13 14 15	◢	NL	(0A)		NL	(0A)		NL	(5B)		LF-CR	(5B)		NL	(5B)		LF-CR	(5B)
16 17 18 19																		
20 21 22 23	◢	NL	0A		NL	(0A)		NL BS IL	5B [DB] 5D [DD] 5E [DE]		RES LF-CR BS IL	58 [D8] 5B [DB] 5D [DD] 5E [DE]		RES NL BS IL	58 [D8] 5B [DB] 5D [DD] 5E [DE]		RES LF-CR BS IL	58 5B [DB] 5D [DD] 5E
24 25 26 27	▮	CAN Check	[18] 42	"		(42)					PRE	3E						
28 29 30 31																		
32 33 34 35																		
36 37 38 39		ETX	(03)		ETX	(03)		Index Attn EOB PRE	3B [BB] 3D [BD] 3E		BYP Index Attn EOB(Ⓑ)	38 3B [BB] [3D] [BD]		BY LF EOB PRE	38 [B8] 3B [BB] [3D] [BD] 3E [BE]		BYP Index Attn EOB(Ⓑ) PRE	38 3B [BB] [3D] 3E
40 41 42 43																		
44 45 46 47		ACK	06		ACK	06												
48 49 50 51								IL	(5E)		IL	(5E)		IL	(5E)			
52 53 54 55		EOT(Ⓒ)	04		EOT(Ⓒ)	04		Upshft EOT(Ⓒ)	1C [9C] 1F		Upshft EOT(Ⓒ)	1C [9C] 1F		RS UC EOT(Ⓒ)	[1A] [9A] 1C [9C] 1F		PN RS Upshft EOT(Ⓒ)	19 1A 1C [9C] 1F [9F]
56 57 58 59																		
60 61 62 63	:	NAK	15 (5A)	:	NAK	[15] (5A)	:		(88)	:		(88)				:		(EB)
64 65 66 67		SP	40		SP	(40)		SP	01 [81]		SP	01 [81]		SP	01 [81]		SP	01 [81]
68 69 70 71																		

Ref.	AT&T 83 B3 W U 115A			AT&T TWX			WT Telegraph: ITA2			WT Telegraph: ZSC3			EBCDIC	Ref.
	Character		Code (Hex)	Character		Code (Hex) Even Non	Character		Code (Hex)	Character		Code (Hex)	Code (Hex)	
	Graphic	Control		Graphic	Control		Graphic	Control		Graphic	Control			
	37	38	39	40	41	42	43	44	45	46	47	48	49	Ref.
0 1 2 3		LTRS CR	(1F) (02)		Rubout CR	(FF) (FF) (B1) (B1)		LTRS CR	(1F) (02)		LTRS CR	(1F) (02)	00 01 02 03	0 1 2 3
4 5 6 7		LTRS LTRS	1F [3F] (1F)		TpAuxOff HT Rubout	28 29 90 91 FF FF		LTRS LTRS	1F [3F] (1F)		LTRS LTRS	1F [3F] (1F)	04 05 06 07	4 5 6 7
8 9 10 11					VT	D1 D1							08 09 0A 0B	8 9 10 11
12 13 14 15		CR	(02) (22)		FF CR SO SI	30 31 (B1) (B1) 71 71 F0 F1		CR	(02)		CR	(02)	0C 0D 0E 0F	12 13 14 15
16 17 18 19					X-On	8B 89							10 11 12 13	16 17 18 19
20 21 22 23		LF LTRS	08 [28] (1F)		LF Rubout	50 [51] (FF) (FF)		CR LTRS	02 [22] (1F)		CR LTRS	02 [22] (1F)	14 15 16 17	20 21 22 23
24 25 26 27													18 19 1A 1B	24 25 26 27
28 29 30 31													1C 1D 1E 1F	28 29 30 31
32 33 34 35													20 21 22 23	32 33 34 35
36 37 38 39		LF CR	(08) 02 [22]		LF CR	(50) (51) B1 [B1]		LF WRU	08 [28] (08) [32]		LF WRU	08 [28] (08) [32]	24 25 26 27	36 37 38 39
40 41 42 43								Bell	3A		Bell	2C	28 29 2A 2B	40 41 42 43
44 45 46 47	'C	Bell A	3A		WRU Bell	A0 [A1] E1 E1							2C 2D 2E 2F	44 45 46 47
48 49 50 51		LTRS	(1F)		Rubout	(FF) (FF)		LTRS	(1F)		LTRS	(1F)	30 31 32 33	48 49 50 51
52 53 54 55	# STOP A C	FIGS	1B [3B] (25)		TpAuxOn EOT	48 (49) [C9] [21]		FIGS LF	[1B] 3B (08)		FIGS LF	[1B] 3B (08)	34 35 36 37	52 53 54 55
56 57 58 59													38 39 3A 3B	56 57 58 59
60 61 62 63	/		(37)	:	X-Off	(C9) (C9) (5C) (5D)	/		(37)	/		(2A)	3C 3D 3E 3F	60 61 62 63
64 65 66 67		SP	04 [24]		SP	05 05		SP	04 [24]		SP	04 [24]	40 41 42 43	64 65 66 67
68 69 70 71													44 45 46 47	68 69 70 71

Figure 40. TCAM Internal and Device Codes (Part 1 of 4)

*C9 and 21 are valid EOT characters for both even parity and non parity transmission.

Ref.	Internal S/360 Code (EBCDIC) Character Graphic 1	Internal S/360 Control 2	Internal S/360 Code (Hex) 3	USASCII Character Graphic 4	USASCII Control 5	USASCII Code (Hex) 6	6-Bit Transcode Character Graphic 7	6-Bit Transcode Control 8	6-Bit Transcode Code (Hex) 9	IBM 1030 Character Graphic 10	IBM 1030 Control 11	IBM 1030 Code (Hex) 12	IBM 1050 Character Graphic 13	IBM 1050 Control 14	IBM 1050 Code (Hex) 15	IBM 1060 Character Graphic 16	IBM 1060 Control 17	IBM 1060 Code (Hex) 18
72 73 74 75	¢ .		48 49 4A 4B	[.		5B 2E	.		0B	.	Ⓨ EOFC	76	¢ .	Ⓨ	A0 76	.	Ⓨ	(76)
76 77 78 79	< (+ |		4C 4D 4E 4F	< (+ ¦		3C 28 2B 21	¤		0C				< (+ |		84 93 E1 B7			
80 81 82 83	&		50 51 52 53	&		26	&		10	$\&_S\ +_H$		61	&		61	+		(61)
84 85 86 87			54 55 56 57															
88 89 90 91	! $		58 59 5A 5B	! $		5D 24	$		1B	$		57	! $		D7 57	$		(57)
92 93 94 95	*) ; ¬		5C 5D 5E 5F	*) ; ~		2A 29 3B 5E	*		1C				*) ; ¬	Ⓨ	90 95 87 F6			
96 97 98 99	- /		60 61 62 63	- /		2D 2F	- /		20 21	- /	Ⓝ	40 23	- /	Ⓝ	40 23	- /	Ⓝ	(40) 23
100 101 102 103			64 65 66 67															
104 105 106 107	,	EOM	68 69 6A 6B	| ,		7C 2C	,		2B	,	Ⓢ	37	,		37	,		(37)
108 109 110 111	% _ > ?		6C 6D 6E 6F	% _ > ?		25 5F 3E 3F	%		2C				% _ > ?	Ⓝ	8B C0 8E A3			
112 113 114 115			70 71 72 73															
116 117 118 119			74 75 76 77															
120 121 122 123	: #	EOA	78 79[3] 7A 7B	[: #		60 3A 23	#		3B	$\#_S\ =_H$	EOA(Ⓓ)	16	: #	EOA(Ⓓ)	88 16	#	EOA(Ⓓ)	16
124 125 126 127	@ ' = "		7C 7D 7E 7F	@ ' = "		40 27 3D 22	@		3C	$@_S\ '_H$		(20)	@ ' = "	EOA(Ⓓ)	20 8D 82 96		Add	(20)
128 129 130 131	a b c		80 81 82 83	a b c		61 62 63	A B C		(01) (02) (03)	A B C		(62) (64) (67)	a b c		62 64 67	A B C		(62) (64) (67)
132 133 134 135	d e f g		84 85 86 87	d e f g		64 65 66 67	D E F G		(04) (05) (06) (07)	D E F G		(68) (6B) (6D) (6E)	d e f g		68 6B 6D 6E	D E F G		(68) (6B) (6D) (6E)
136 137 138 139	h i		88 89 8A 8B	h i		68 69	H I		(08) (09)	H I		(70) (73)	h i		70 73	H I		(70) (73)
140 141 142 143			8C 8D 8E 8F															

Ref.	IBM 2260 (Remote)/2265: 2260/2265 Character Graphic 19	2260/2265 Control 20	2260/2265 Code (Hex) 21	1053 Character Graphic 22	1053 Control 23	1053 Code (Hex) 24	IBM 2740 Character Graphic 25	IBM 2740 Control 26	IBM 2740 Code (Hex) 27	2741 BCD Character Graphic 28	2741 BCD Control 29	2741 BCD Code (Hex) 30	2741 EBCD Graphic 31	2741 EBCD Control 32	2741 EBCD Code (Hex) 33	2741 Correspondence Character Graphic 34	2741 Correspondence Control 35	2741 Correspondence Code (Hex) 36
72 73 74 75	▶ .	SMI	BD 4E	.	¢	(BD) (4E)	¢ .	Ⓨ	A0 76	¢ .	Ⓨ	A0 76 [F6]	¢ .	Ⓨ	A0 76	.		45 [C5]
76 77 78 79	< (+ |		5C 48 4B FE	< (+ |		(5C) (48) (4B) (FE)	< (+ |	Ⓢ	84 93 E1 B7	(+ ±	EOA(Ⓓ)	93 E1 96	> (+ |	Ⓢ	84 93 E1 B7	¢ (+ [±		8D 96 E4 82
80 81 82 83	&		46	&		(46)	&		61	&		61	&		61	&		8B
84 85 86 87																		
88 89 90 91	$		44	$		(44)	! $		D7 57	! $		D7 57	! $		D7 57	! $		40 90
92 93 94 95	*) ; ¬		4A 49 5B FC	*) ; ¬		(4A) (49) (5B) (FC)	*) ; ¬	Ⓨ	90 95 87 F6	*) ; °		90 95 87 84	*) ; ¬	Ⓨ	90 95 87 F6	*) ;		8E 93 6B
96 97 98 99	- /		4D 4F	- /		(4D) (4F)	- /	Ⓝ	40 23	- /	Ⓝ	40 23	- /	Ⓝ	40 23	- /		76 70
100 101 102 103																		
104 105 106 107	▬ ,	EOM	41 4C	! ,		(41) (4C)	,	Ⓢ	37	,	Ⓢ	37 [B7]	,	Ⓢ	37	,		[6E] EE
108 109 110 111	% _ > ?		45 BF 5E 5F	% _ > ?		(45) (BF) (5E) (5F)	% _ > ?	Ⓝ	8B C0 8E A3	% _ " ?	Ⓝ	8B C0 8E A3	% - > ?	Ⓝ	8B C0 8E A3	% _ ° ?		88 F6 C0 F0
112 113 114 115																		
116 117 118 119																		
120 121 122 123	: #		5A 43	: #		(5A) (43)	: #	EOA(Ⓓ)	88 16	: #	EOA(Ⓓ)	88 16	: #	EOA(Ⓓ)	88 16	: #		EB 87
124 125 126 127	@ ' =		E0 47 5D	@ ' =		(E0) (47) (5D)	@ ' = "	EOA	20 8D 82 96	@ ' = "		20 8D 82 8E	@ ' = "	EOA	20 8D 82 96	@ ' = "		84 49 64 C9
128 129 130 131	A B C		(A1) (A2) (A3)	A B C		(A1) (A2) (A3)	a b c		62 64 67	a b c		62 64 67	a b c		62 64 67	a b c		4F 37 2F
132 133 134 135	D E F G		(A4) (A5) (A6) (A7)	D E F G		(A4) (A5) (A6) (A7)	d e f g		68 6B 6D 6E	d e f g		68 6B 6D 6E	d e f g		68 6B 6D 6E	d e f g		2A 29 67 62
136 137 138 139	H I		(A8) (A9)	H I		(A8) (A9)	h i		70 73	h i		70 73	h i		70 73	h i		32 4C
140 141 142 143																		

Ref.	AT&T 83B3 W U 115A Character Graphic 37	AT&T 83B3 Control 38	AT&T 83B3 Code (Hex) 39	AT&T TWX Character Graphic 40	AT&T TWX Control 41	AT&T TWX Code (Hex) 42	WT Telegraph ITA2 Character Graphic 43	ITA2 Control 44	ITA2 Code (Hex) 45	ZSC3 Character Graphic 46	ZSC3 Control 47	ZSC3 Code (Hex) 48	EBCDIC Code (Hex) 49	Ref.
72 73 74 75	.		27	] .		BB BB 74 75	.		27	.		21	48 49 4A 4B	72 73 74 75
76 77 78 79	$(_A\ 1/2_C$		3E	< (+ ↑		3C 3D 14 15 D4 D5 7B 7B	(+		3E 31	(+		3E 38	4C 4D 4E 4F	76 77 78 79
80 81 82 83	&		2B	&		65 65							50 51 52 53	80 81 82 83
84 85 86 87													54 55 56 57	84 85 86 87
88 89 90 91	$2_A\ 1/4_C$ $		36 32	! $		84 85 24 25							58 59 5A 5B	88 89 90 91
92 93 94 95	$)_A\ 3/4_C$ $;_A\ 3/8_C$		29 2F	*) ;		55 55 95 95 DD DD	)		29	)		29	5C 5D 5E 5F	92 93 94 95
96 97 98 99	- /		38 37	- /		B4 B5 F5 F5	- /		38 37	- /		30 2A	60 61 62 63	96 97 98 99
100 101 102 103													64 65 66 67	100 101 102 103
104 105 106 107	$,_A\ 7/8_C$		26	,		35 35	,		26	,		26	68 69 6A 6B	104 105 106 107
108 109 110 111	$?_A\ 5/8_C$		33	% ← > ?		A5 A5 FA FB 7D 7D FC FD	?		33	?		25	6C 6D 6E 6F	108 109 110 111
112 113 114 115													70 71 72 73	112 113 114 115
116 117 118 119													74 75 76 77	116 117 118 119
120 121 122 123	$:_A\ 1/8_C$	CR	2E (02)	[: #		DB DB 5C 5D C5 C5	:		2E	:		23	78 79 7A 7B	120 121 122 123
124 125 126 127	$'_A$ "	$Bell_C$	34 31	@ ' = "		03 03 E4 E5 BD BD 44 45	' =		34 2F	' =		34 2F	7C 7D 7E 7F	124 125 126 127
128 129 130 131	A B C		(18) (13) (0E)	A B C		(82) (83) (42) (43) (C3) (C3)	A B C		(18) (13) (0E)	A B C		(18) (13) (0E)	80 81 82 83	128 129 130 131
132 133 134 135	D E F G		(12) (10) (16) (0B)	D E F G		(22) (23) (A3) (A3) (63) (63) (E2) (E3)	D E F G		(12) (10) (16) (0B)	D E F G		(12) (10) (16) (0B)	84 85 86 87	132 133 134 135
136 137 138 139	H I		(05) (0C)	H I		(12) (13) (93) (93)	H I		(05) (0C)	H I		(05) (0C)	88 89 8A 8B	136 137 138 139
140 141 142 143													8C 8D 8E 8F	140 141 142 143

Figure 40. TCAM Internal and Device Codes (Part 2 of 4)

Ref.	Internal S/360 Code (EBCDIC) Character Graphic	Internal S/360 Code (EBCDIC) Character Control	Internal S/360 Code (EBCDIC) Code (Hex)	USASCII Character Graphic	USASCII Character Control	USASCII Code (Hex)	6-Bit Transcode Character Graphic	6-Bit Transcode Character Control	6-Bit Transcode Code (Hex)	IBM 1030 Character Graphic	IBM 1030 Character Control	IBM 1030 Code (Hex)	IBM 1050 Character Graphic	IBM 1050 Character Control	IBM 1050 Code (Hex)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
144			90												
145	j		91	j		6A	J		(11)	J		(43)	j		43
146	k		92	k		6B	K		(12)	K		(45)	k		45
147	l		93	l		6C	L		(13)	L		(46)	l		46
148	m		94	m		6D	M		(14)	M		(49)	m		49
149	n		95	n		6E	N		(15)	N		(4A)	n		4A
150	o		96	o		6F	O		(16)	O		(4C)	o		4C
151	p		97	p		70	P		(17)	P		(4F)	p		4F
152	q		98	q		71	Q		(18)	Q		(51)	q		51
153	r		99	r		72	R		(19)	R		(52)	r		52
154			9A												
155			9B												
156			9C												
157			9D												
158			9E												
159			9F												
160			A0												
161			A1[2]	~		7E									
162	s		A2	s		73	S		(22)	S		(25)	s		25
163	t		A3	t		74	T		(23)	T		(26)	t		26
164	u		A4	u		75	U		(24)	U		(29)	u		29
165	v		A5	v		76	V		(25)	V		(2A)	v		2A
166	w		A6	w		77	W		(26)	W		(2C)	w		2C
167	x		A7	x		78	X		(27)	X		(2F)	x		2F
168	y		A8	y		79	Y		(28)	Y		(31)	y		31
169	z		A9	z		7A	Z		(29)	Z		(32)	z		32
170			AA												
171			AB												
172			AC												
173			AD												
174			AE												
175			AF												
176			B0												
177			B1												
178			B2												
179			B3												
180			B4												
181			B5												
182			B6												
183			B7												
184			B8												
185			B9												
186			BA												
187			BB												
188			BC												
189			BD												
190			BE												
191			BF												
192		PZ	C0	{		7B								PZ	75
193	A		C1	A		41	A		01	A		62	A		E2
194	B		C2	B		42	B		02	B		64	B		E4
195	C		C3	C		43	C		03	C		67	C		E7
196	D		C4	D		44	D		04	D		68	D		E8
197	E		C5	E		45	E		05	E		6B	E		EB
198	F		C6	F		46	F		06	F		6D	F		ED
199	G		C7	G		47	G		07	G		6E	G		EE
200	H		C8	H		48	H		08	H		70	H		F0
201	I		C9	I		49	I		09	I		73	I		F3
202			CA												
203			CB												
204			CC												
205			CD												
206			CE												
207			CF	\		(5C)									
208		MZ	D0	}		7D								MZ	54
209	J		D1	J		4A	J		11	J		43	J		C3
210	K		D2	K		4B	K		12	K		45	K		C5
211	L		D3	L		4C	L		13	L		46	L		C6
212	M		D4	M		4D	M		14	M		49	M		C9
213	N		D5	N		4E	N		15	N		4A	N		CA
214	O		D6	O		4F	O		16	O		4C	O		CC
215	P		D7	P		50	P		17	P		4F	P		CF

Ref.	IBM 1060 Character Graphic	IBM 1060 Character Control	IBM 1060 Code (Hex)	IBM 2260 (Remote)/2265 — 2260/2265 Character Graphic	2260/2265 Character Control	2260/2265 Code (Hex)	1053 Character Graphic	1053 Character Control	1053 Code (Hex)	IBM 2740 Character Graphic	IBM 2740 Character Control	IBM 2740 Code (Hex)	2741 — BCD Character Graphic	BCD Character Control	BCD Code (Hex)	EBCD Character Graphic	EBCD Character Control	EBCD Code (Hex)
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
144																		
145	J		(43)	J		(AA)	J		(AA)	j		43	j		43	j		43
146	K		(45)	K		(AB)	K		(AB)	k		45	k		45	k		45
147	L		(46)	L		(AC)	L		(AC)	l		46	l		46	l		46
148	M		(49)	M		(AD)	M		(AD)	m		49	m		49	m		49
149	N		(4A)	N		(AE)	N		(AE)	n		4A	n		4A	n		4A
150	O		(4C)	O		(AF)	O		(AF)	o		4C	o		4C	o		4C
151	P		(4F)	P		(B0)	P		(B0)	p		4F	p		4F	p		4F
152	Q		(51)	Q		(B1)	Q		(B1)	q		51	q		51	q		51
153	R		(52)	R		(B2)	R		(B2)	r		52	r		52	r		52
154																		
155																		
156																		
157																		
158																		
159																		
160																		
161																		
162	S		(25)	S		(B3)	S		(B3)	s		25	s		25	s		25
163	T		(26)	T		(B4)	T		(B4)	t		26	t		26	t		26
164	U		(29)	U		(B5)	U		(B5)	u		29	u		29	u		29
165	V		(2A)	V		(B6)	V		(B6)	v		2A	v		2A	v		2A
166	W		(2C)	W		(B7)	W		(B7)	w		2C	w		2C	w		2C
167	X		(2F)	X		(B8)	X		(B8)	x		2F	x		2F	x		2F
168	Y		(31)	Y		(B9)	Y		(B9)	y		31	y		31	y		31
169	Z		(32)	Z		(BA)	Z		(BA)	z		32	y		32	z		32
170																		
171																		
172																		
173																		
174																		
175																		
176																		
177																		
178																		
179																		
180																		
181																		
182																		
183																		
184																		
185																		
186																		
187																		
188																		
189																		
190																		
191																		
192		Restore	(75)															
193	A		62	A		A1	A		(A1)	A		E2	A		E2	A		E2
194	B		64	B		A2	B		(A2)	B		E4	B		E4	B		E4
195	C		(67)	C		A3	C		(A3)	C		E7	C		E7	C		E7
196	D		(68)	D		A4	D		(A4)	D		E8	D		E8	D		E8
197	E		6B	E		A5	E		(A5)	E		EB	E		EB	E		EB
198	F		6D	F		A6	F		(A6)	F		ED	F		ED	F		ED
199	G		(6E)	G		A7	G		(A7)	G		EE	G		EE	G		EE
200	H		(70)	H		A8	H		(A8)	H		F0	H		F0	H		F0
201	I		73	I		A9	I		(A9)	I		F3	I		F3	I		F3
202																		
203																		
204																		
205																		
206																		
207																		
208		Message	(54)															
209	J		43	J		AA	J		(AA)	J		C3	J		C3	J		C3
210	K		45	K		AB	K		(AB)	K		C5	K		C5	K		C5
211	L		(46)	L		AC	L		(AC)	L		C6	L		C6	L		C6
212	M		(49)	M		AD	M		(AD)	M		C9	M		C9	M		C9
213	N		(4A)	N		AE	N		(AE)	N		CA	N		CA	N		CA
214	O		(4C)	O		AF	O		(AF)	O		CC	O		CC	O		CC
215	P		(4F)	P		B0	P		(B0)	P		CF	P		CF	P		CF

Ref.	2741 Correspondence Character Graphic	Correspondence Character Control	Correspondence Code (Hex)	AT&T 83B3 W U 115A Character Graphic	AT&T 83B3 W U 115A Character Control	AT&T 83B3 W U 115A Code (Hex)	AT&T TWX Character Graphic	AT&T TWX Character Control	AT&T TWX Code (Hex)	WT Telegraph — ITA2 Character Graphic	ITA2 Character Control	ITA2 Code (Hex)	ZSC3 Character Graphic	ZSC3 Character Control	ZSC3 Code (Hex)	EBCDIC Code (Hex)	Re.
	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	
144																90	144
145	j		61	J		(1A)	J		(53) (53)	J		(1A)	J		(1A)	91	145
146	k		2C	K		(1E)	K		(D2) (D3)	K		(1E)	K		(1E)	92	146
147	l		31	L		(09)	L		(33) (33)	L		(09)	L		(09)	93	147
148	m		43	M		(07)	M		(B2) (B3)	M		(07)	M		(07)	94	148
149	n		25	N		(06)	N		(72) (73)	N		(06)	N		(06)	95	149
150	o		51	O		(03)	O		(F3) (F3)	O		(03)	O		(03)	96	150
151	p		68	P		(0D)	P		(0A) (0B)	P		(0D)	P		(0D)	97	151
152	q		6D	Q		(1D)	Q		(8B) (8B)	Q		(1D)	Q		(1D)	98	152
153	r		4A	R		(0A)	R		(4B) (4B)	R		(0A)	R		(0A)	99	153
154																9A	154
155																9B	155
156																9C	156
157																9D	157
158																9E	158
159																9F	159
160																A0	160
161																A1	161
162	s		52	S		(14)	S		(CA) (CB)	S		(14)	S		(14)	A2	162
163	t		20	T		(01)	T		(2B) (2B)	T		(01)	T		(01)	A3	163
164	u		26	U		(1C)	U		(AA) (AB)	U		(1C)	U		(1C)	A4	164
165	v		46	V		(0F)	V		(6A) (6B)	V		(0F)	V		(0F)	A5	165
166	w		57	W		(19)	W		(EB) (EB)	W		(19)	W		(19)	A6	166
167	x		23	X		(17)	X		(1B) (1B)	X		(17)	X		(17)	A7	167
168	y		73	Y		(15)	Y		(9A) (9B)	Y		(15)	Y		(15)	A8	168
169	z		15	Z		(11)	Z		(5A) (5B)	Z		(11)	Z		(11)	A9	169
170																AA	170
171																AB	171
172																AC	172
173																AD	173
174																AE	174
175																AF	175
176																B0	176
177																B1	177
178																B2	178
179																B3	179
180																B4	180
181																B5	181
182																B6	182
183																B7	183
184																B8	184
185																B9	185
186																BA	186
187																BB	187
188																BC	188
189																BD	189
190																BE	190
191																BF	191
192																C0	192
193	A		CF	A		18	A		82 83	A		18	A		18	C1	193
194	B		B7	B		13	B		42 43	B		13	B		13	C2	194
195	C		AF	C		0E	C		C3 C3	C		0E	C		0E	C3	195
196	D		AA	D		12	D		22 23	D		12	D		12	C4	196
197	E		A9	E		10	E		A3 A3	E		10	E		10	C5	197
198	F		E7	F		16	F		63 63	F		16	F		16	C6	198
199	G		E2	G		0B	G		E2 E3	G		0B	G		0B	C7	199
200	H		B2	H		05	H		12 13	H		05	H		05	C8	200
201	I		CC	I		0C	I		93 93	I		0C	I		0C	C9	201
202																CA	202
203																CB	203
204																CC	204
205																CD	205
206																CE	206
207																CF	207
208																D0	208
209	J		E1	J		1A	J		53 53	J		1A	J		1A	D1	209
210	K		AC	K		1E	K		D2 D3	K		1E	K		1E	D2	210
211	L		B1	L		09	L		33 33	L		09	L		09	D3	211
212	M		C3	M		07	M		B2 B3	M		07	M		07	D4	212
213	N		A5	N		06	N		72 73	N		06	N		06	D5	213
214	O		D1	O		03	O		F3 F3	O		03	O		03	D6	214
215	P		E8	P		0D	P		0A 0B	P		0D	P		0D	D7	215

2. No EBCDIC character has been assigned to this location (161/3, 161/4).

Figure 40.. TCAM Internal and Device Codes (Part 3 of 4)

Ref.	Internal S/360 Code (EBCDIC) Character 1	Internal S/360 Code (EBCDIC) Character 2	Internal S/360 Code (EBCDIC) Code (Hex) 3	USASCII Character 4	USASCII Character 5	USASCII Code (Hex) 6	6-Bit Transcode Character 7	6-Bit Transcode Character 8	6-Bit Transcode Code (Hex) 9	IBM 1030 Character 10	IBM 1030 Character 11	IBM 1030 Code (Hex) 12	IBM 1050 Character 13	IBM 1050 Character 14	IBM 1050 Code (Hex) 15	IBM 1060 Character 16	IBM 1060 Character 17	IBM 1060 Code (Hex) 18	IBM 2260 (Remote) /2265 2260/2265 Character 19	IBM 2260 (Remote) /2265 2260/2265 Control 20	IBM 2260 (Remote) /2265 2260/2265 Code (Hex) 21	IBM 2260 (Remote) /2265 1053 Character 22	IBM 2260 (Remote) /2265 1053 Character 23	IBM 2260 (Remote) /2265 1053 Code (Hex) 24	IBM 2740 Character 25	IBM 2740 Character 26	IBM 2740 Code (Hex) 27	2741 BCD Character 28	2741 BCD Character 29	2741 BCD Code (Hex) 30	2741 EBCD Character 31	2741 EBCD Character 32	2741 EBCD Code (Hex) 33	2741 Correspondence Character 34	2741 Correspondence Character 35	2741 Correspondence Code (Hex) 36	AT&T 83B3 W U 115A Character 37	AT&T 83B3 W U 115A Character 38	AT&T 83B3 W U 115A Code (Hex) 39	AT&T TWX Character 40	AT&T TWX Character 41	AT&T TWX Code (Hex) 42	WT Telegraph ITA2 Character 43	WT Telegraph ITA2 Character 44	WT Telegraph ITA2 Code (Hex) 45	WT Telegraph Z5C3 Character 46	WT Telegraph Z5C3 Character 47	WT Telegraph Z5C3 Code (Hex) 48	EBCDIC Code (Hex) 49	Ref.
216	Q		D8	Q		51	Q		18	Q		51	Q		D1	Q		(51)	Q		B1	Q		(81)	Q		D1	Q		D1	Q		D1	Q		ED	Q		1D	Q		1D	Q		1D	D8	216
217	R		D9	R		52	R		19	R		52	R		D2	R		(52)	R		B2	R		(82)	R		D2	R		D2	R		D2	R		CA	R		0A	R		0A	R		0A	D9	217
218			DA																																									DA	218
219			DB																																									DB	219
220			DC																																									DC	220
221			DD																																									DD	221
222			DE																																									DE	222
223			DF																																									DF	223
224		RM	E0[3]	\		5C							‡	RM	34																												E0	224
225			E1												A5																						/							E1	225
226	S		E2	S		53	S		22	S		25	S		A6	S		(25)	S		B3	S		(83)	S		A5	S		A5	S		A5	S		D2	S		14	S		14	E2	226
227	T		E3	T		54	T		23	T		26	T			T		(26)	T		B4	T		(84)	T		A6	T		A6	T		A6	T		A0	T		01	T		01	E3	227
228	U		E4	U		55	U		24	U		29	U		A9	U		(29)	U		B5	U		(85)	U		A9	U		A9	U		A9	U		A6	U		1C	U		1C	E4	228
229	V		E5	V		56	V		25	V		2A	V		AA	V		(2A)	V		B6	V		(86)	V		AA	V		AA	V		AA	V		C6	V		0F	V		0F	E5	229
230	W		E6	W		57	W		26	W		2C	W		AC	W		(2C)	W		B7	W		(87)	W		AC	W		AC	W		AC	W		D7	W		19	W		19	E6	230
231	X		E7	X		58	X		27	X		2F	X		AF	X		(2F)	X		B8	X		(88)	X		AF	X		AF	X		AF	X		A3	X		17	X		17	E7	231
232	Y		E8	Y		59	Y		28	Y		31	Y		B1	Y		(31)	Y		B9	Y		(89)	Y		B1	Y		B1	Y		B1	Y		F3	Y		15	Y		15	E8	232
233	Z		E9	Z		5A	Z		29	Z		32	Z		B2	Z		(32)	Z		BA	Z		(8A)	Z		B2	Z		B2	Z		B2	Z		95	Z		11	Z		11	E9	233
234			EA																																								EA	234
235			EB																																								EB	235
236			EC																																								EC	236
237			ED																																								ED	237
238			EE																																								EE	238
239			EF																																								EF	239
240	0		F0	0		30	0		30	0[4]		(15) [20]	0		15	0		15	0		50	0		(50)	0		15	0		15	0		15	0		13	0		2D	0		2D	0		2B	F0	240
241	1		F1	1		31	1		31	1		02	1		02	1		02	1		51	1		(51)	1		02	1		02	1		02	1		02	1		3D	1		3D	3		3C	F1	241
242	2		F2	2		32	2		32	2		04	2		04	2		04	2		52	2		(52)	2		04	2		04	2		04	2		04	2		39	2		39	A		3A	F2	242
243	3		F3	3		33	3		33	3		07	3		07	3		07	3		53	3		(53)	3		07	3		07	3		07	3		07	3		30	3		30	B		3B	F3	243
244	4		F4	4		34	4		34	4		08	4		08	4		08	4		54	4		(54)	4		08	4		08	4		08	4		10	4		2A	4		2A	4		36	F4	244
245	5		F5	5		35	5		35	5		0B	5		0B	5		0B	5		55	5		(55)	5		0B	5		0B	5		0B	5		08	5		21	5		35	5		35	F5	245
246	6		F6	6		36	6		36	6		0D	6		0D	6		0D	6		56	6		(56)	6		0D	6		0D	6		0D	6		0D	6		35	6		33	6		33	F6	246
247	7		F7	7		37	7		37	7		0E	7		0E	7		0E	7		57	7		(57)	7		0E	7		0E	7		0E	7		08	7		3C	7		27	7		27	F7	247
248	8		F8	8		38	8		38	8		10	8		10	8		10	8		58	8		(58)	8		10	8		10	8		10	8		0E	8		2C	8		2E	8		2E	F8	248
249	9		F9	9		39	9		39	9		13	9		13	9		13	9		59	9		(59)	9		13	9		13	9		13	9	EOA	16	9		23	9		2D	9		2D	F9	249
250			FA																																									FA	250
251			FB																																										FB	251
252			FC																																										FC	252
253			FD																																										FD	253
254			FE																																										FE	254
255			FF																																										FF	255

3. No EBCDIC character has been assigned to this location (225/3, 225/37).

Figure 40. TCAM Internal and Device Codes (Part 4 of 4)

S/360 Byte (hex)	Graphic	Control
00		SOH
01	A	
02	B	
03	C	
04	D	
05	E	
06	F	
07	G	
08	H	
09	I	
0A		STX
0B		
0C	¤	
0D		BEL
0E		SUB
0F		ETB
10	&	
11	J	
12	K	
13	L	
14	M	
15	N	
16	O	
17	P	
18	Q	
19	R	
1A		SP
1B	$	
1C	*	
1D		US
1E		EOT
1F		DLE
20	-	
21	/	
22	S	
23	T	
24	U	
25	V	
26	W	
27	X	
28	Y	
29	Z	
2A		ESC
2B	,	
2C	%	
2D		ENQ
2E		ETX
2F		HT
30	0	
31	1	
32	2	
33	3	
34	4	
35	5	
36	6	
37	7	
38	8	
39	9	
3A		SYN
3B	#	
3C	@	
3D		NAK
3E		EM
3F		DEL

S/360 Byte (hex)	Graphic	Control
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
4A		
4B		
4C		
4D		
4E		
4F		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
5A		
5B		
5C		
5D		
5E		
5F		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
6A		
6B		
6C		
6D		
6E		
6F		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
7A		
7B		
7C		
7D		
7E		
7F		

S/360 Byte (hex)	Graphic	Control
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
CD		
CE		
CF		
D0		
D1		
D2		
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		
FB		
FC		
FD		
FE		
FF		

Figure 43. Hexadecimal Equivalents for 6-bit Transcode

S/360 Byte (hex)	Graphic	Control
00		
01		SP
02	1	
03		
04	2	
05		
06		
07	3	
08	4	
09		
0A		
0B	5	
0C		
0D	6	
0E	7	
0F		
10	8	
11		
12		
13	9	
14	0^1 @	
15		
16	$\#_S$ $=_H$	EOA
17		
18		
19		
1A		
1B		
1C		
1D		
1E		
1F		EOT
20	0_S^1 1_H	
21		
22		
23	/	
24		
25	S	
26	T	
27		
28		
29	U	
2A	V	
2B		
2C	W	
2D		
2E		
2F	X	
30		
31	Y	
32	Z	
33		
34		
35		
36		
37	,	(S)
38		
39		
3A		
3B		LF
3C		
3D		EOB
3E		
3F		

S/360 Byte (hex)	Graphic	Control
40	-	(N)
41		
42		
43	J	
44		
45	K	
46	L	
47		
48		
49	M	
4A	N	
4B		
4C	O	
4D		
4E		
4F	P	
50		
51	Q	
52		
53	R	
54		
55		
56		
57	$	
58		
59		
5A		
5B		LF-CR
5C		
5D		
5E		
5F		
60		
61	$\&_S$ $+_H$	
62	A	
63		
64	B	
65		
66		
67	C	
68	D	
69		
6A		
6B	E	
6C		
6D	F	
6E	G	
6F		
70	H	
71		
72		
73	I	
74		
75		
76		(Y) EOFC[2]
77		
78		
79		
7A		HT
7B		
7C		
7D		
7E		
7F		EOC DEL

S/360 Byte (hex)	Graphic	Control
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
CD		
CE		
CF		
D0		
D1		
D2		
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		
FB		
FC		
FD		
FE		
FF		

Note 1: The IBM 1031 Input Station transmits the numeric 0 as an A bit only; the IBM 1033 Printer receives a numeric 0 as C-8-2 and an @ as an A bit.

Note 2: The IBM 1031 Input Station cannot transmit the following characters as data: % * . @ ¤ (an EOFC is transmitted and punched by the IBM 1034 Card Punch).

Figure 44. Line Code for IBM 1030 Data Collection System

S/360 Byte (hex)	Graphic	Control
00		
01		SP
02	1	
03		
04	2	
05		
06		
07	3	
08	4	
09		
0A		
0B	5	
0C		
0D	6	
0E	7	
0F		
10	8	
11		
12		
13	9	
14		
15	0	
16	#	EOA
17		
18		
19		PN
1A		RS
1B		
1C		Upshift
1D		
1E		
1F		EOT
20	@	
21		
22		
23	/	
24		
25	s	
26	t	
27		
28		
29	u	
2A	v	
2B		
2C	w	
2D		
2E		
2F	x	
30		
31	y	
32	z	
33		
34	¢	RM
35		
36		
37	,	
38		BYP
39		
3A		
3B		LF
3C		
3D		EOB
3E		PRE
3F		

S/360 Byte (hex)	Graphic	Control
40	-	Ⓝ
41		
42		
43	¡	
44		
45	k	
46	l	
47		
48		
49	m	
4A	n	
4B		
4C	o	
4D		
4E		
4F	p	
50		
51	q	
52	r	
53		
54		MZ
55		
56		
57	$	
58		RES
59		
5A		
5B		NL
5C		
5D		BS
5E		IL
5F		
60		
61	&	
62	a	
63		
64	b	
65		
66		
67	c	
68	d	
69		
6A		
6B	e	
6C		
6D	f	
6E	g	
6F		
70	h	
71		
72		
73	i	
74		
75		PZ
76	.	Ⓨ
77		
78		
79		PF
7A		TAB
7B		
7C		Dwnshft
7D		
7E		
7F		DEL

S/360 Byte (hex)	Graphic	Control
80		
81		SP
82	=	
83		
84	<	
85		
86		
87	;	
88	:	
89		
8A		
8B	%	
8C		
8D	'	
8E	>	
8F		
90	*	
91		
92		
93	(	
94		
95	)	
96	"	EOA
97		
98		
99		PN
9A		RS
9B		
9C		Upshift
9D		
9E		
9F		
A0	¢	
A1		
A2		
A3	?	
A4		
A5	S	
A6	T	
A7		
A8		
A9	U	
AA	V	
AB		
AC	W	
AD		
AE		
AF	X	
B0		
B1	Y	
B2	Z	
B3		
B4		
B5		
B6		
B7		
B8		BYP
B9		
BA		
BB		LF
BC		
BD		EOB
BE		PRE
BF		

S/360 Byte (hex)	Graphic	Control
C0	-	Ⓝ
C1		
C2		
C3	J	
C4		
C5	K	
C6	L	
C7		
C8		
C9	M	
CA	N	
CB		
CC	O	
CD		
CE		
CF	P	
D0		
D1	Q	
D2	R	
D3		
D4		
D5		
D6		
D7	!	
D8		RES
D9		
DA		
DB		NL
DC		
DD		BS
DE		IL
DF		
E0		
E1	+	
E2	A	
E3		
E4	B	
E5		
E6		
E7	C	
E8	D	
E9		
EA		
EB	E	
EC		
ED	F	
EE	G	
EF		
F0	H	
F1		
F2		
F3	I	
F4		
F5		
F6	¬	Ⓨ
F7		
F8		
F9		PF
FA		TAB
FB		
FC		Dwnshft
FD		
FE		
FF		DEL

Figure 45. Line Code for IBM 1050 Data Communication System

S/360 Byte (hex)	Graphic	Control
00		
01		SP
02	1	
03		
04	2	
05		
06		
07	3	
08	4	
09		
0A		
0B	5	
0C		
0D	6	
0E	7	
0F		
10	8	
11		
12		
13	9	
14		
15	0	
16	#	EOA
17		
18		
19		
1A		
1B		
1C		
1D		
1E		
1F		EOT
20		Add
21		
22		
23	/	
24		
25	S	
26	T	
27		
28		
29	U	
2A	V	
2B		
2C	W	
2D		
2E		
2F	X	
30		
31	Y	
32	Z	
33		
34		
35		
36		
37	,	
38		
39		
3A		
3B		LF
3C		
3D		EOB
3E		
3F		

S/360 Byte (hex)	Graphic	Control
40	-	Ⓝ
41		
42		
43	J	
44		
45	K	
46	L	
47		
48		
49	M	
4A	N	
4B		
4C	O	
4D		
4E		
4F	P	
50		
51	Q	
52	R	
53		
54		Message
55		
56		
57	$	
58	*	
59		
5A		
5B		CR
5C		
5D		
5E		IL
5F		
60		
61	+	
62	A	
63		
64	B	
65		
66		
67	C	
68	D	
69		
6A		
6B	E	
6C		
6D	F	
6E	G	
6F		
70	H	
71		
72		
73	I	
74		
75		Restore
76	.	Ⓨ
77		
78		
79		Subtr
7A		Tab
7B		
7C		
7D		
7E		
7F		DEL

S/360 Byte (hex)	Graphic	Control
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
CD		
CE		
CF		
D0		
D1		
D2		
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		
FB		
FC		
FD		
FE		
FF		

Figure 46. Line Code for IBM 1060 Data Communication System

S/360 Byte (hex)	2260/2265 Graphic	2260/2265 Control	1053 Graphic	1053 Control
00				
01				
02		STX		STX
03		ETX		ETX
04		EOT		EOT
05				
06		ACK		ACK
07				
08				
09				
0A	◢	NL		NL
0B				
0C				
0D				
0E				
0F				
10				
11				
12				
13				
14				
15		NAK		NAK
16				
17				
18		CAN		
19				
1A				
1B				
1C				
1D				
1E				
1F				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
2A				
2B				
2C				
2D				
2E				
2F				
30				
31				
32				
33				
34				
35				
36				
37				
38				
39				
3A				
3B				
3C				
3D				
3E				
3F				

S/360 Byte (hex)	2260/2265 Graphic	2260/2265 Control	1053 Graphic	1053 Control
40		SP		SP
41	▬	EOM	!	
42	▮	CHECK	"	
43	▯		#	
44	$		$	
45	%		%	
46	&		&	
47	'		'	
48	(		(	
49	)		)	
4A	*		*	
4B	+		+	
4C	,		,	
4D	-		-	
4E	.		.	
4F	/		/	
50	0		0	
51	1		1	
52	2		2	
53	3		3	
54	4		4	
55	5		5	
56	6		6	
57	7		7	
58	8		8	
59	9		9	
5A	:		:	
5B	;		;	
5C	<		<	
5D	=		=	
5E	>		>	
5F	?		?	
60				
61				
62				
63				
64				
65				
66				
67				
68				
69				
6A				
6B				
6C				
6D				
6E				
6F				
70				
71				
72				
73				
74				
75				
76				
77				
78				
79				
7A				
7B				
7C				
7D				
7E				
7F				

Figure 47. Line Codes for IBM 2260 (Remote)/2265 Display Complexes and IBM 1053 Printer (Part 1 of 2)

S/360 Byte (hex)	2260/2265 Graphic	2260/2265 Control	1053 Graphic	1053 Control
80				
81				
82				
83				
84				
85				
86				
87				
88				
89				
8A				
8B				
8C				
8D				
8E				
8F				
90				
91				
92				
93				
94				
95				
96				
97				
98				
99				
9A				
9B				
9C				
9D				
9E				
9F				
A0				
A1	A		A	
A2	B		B	
A3	C		C	
A4	D		D	
A5	E		E	
A6	F		F	
A7	G		G	
A8	H		H	
A9	I		I	
AA	J		J	
AB	K		K	
AC	L		L	
AD	M		M	
AE	N		N	
AF	O		O	
B0	P		P	
B1	Q		Q	
B2	R		R	
B3	S		S	
B4	T		T	
B5	U		U	
B6	V		V	
B7	W		W	
B8	X		X	
B9	Y		Y	
BA	Z		Z	
BB				
BC				
BD	▶	SMI	¢	
BE				
BF	-		-	

S/360 Byte (hex)	2260/2265 Graphic	2260/2265 Control	1053 Graphic	1053 Control
C0				
C1				
C2				
C3				
C4				
C5				
C6				
C7				
C8				
C9				
CA				
CB				
CC				
CD				
CE				
CF				
D0				
D1				
D2				
D3				
D4				
D5				
D6				
D7				
D8				
D9				
DA				
DB				
DC				
DD				
DE				
DF				
E0	@		@	
E1				
E2				
E3				
E4				
E5				
E6				
E7				
E8				
E9				
EA				
EB				
EC				
ED				
EE				
EF				
F0				
F1				
F2				
F3				
F4				
F5				
F6				
F7				
F8				
F9				
FA				
FB				
FC	¬		¬	
FD				
FE	'		'	
FF				

Figure 47. Line Codes for IBM 2260 (Remote)/2265 Display Complexes and IBM 1053 Printer (Part 2 of 2)

S/360 Byte (hex)	Graphic	Control
00		
01		SP
02	1	
03		
04	2	
05		
06		
07	3	
08	4	
09		
0A		
0B	5	
0C		
0D	6	
0E	7	
0F		
10	8	
11		
12		
13	9	
14		
15	0	
16	#	EOA
17		
18		
19		
1A		
1B		
1C		Upshift
1D		
1E		
1F		EOT
20	@	
21		
22		
23	/	
24		
25	s	
26	t	
27		
28		
29	u	
2A	v	
2B		
2C	w	
2D		
2E		
2F	x	
30		
31	y	
32	z	
33		
34		
35		
36		
37	/	Ⓢ
38		
39		
3A		
3B		Index Attn
3C		
3D		EOB
3E		PRE
3F		

S/360 Byte (hex)	Graphic	Control
40	-	Ⓝ
41		
42		
43	j	
44		
45	k	
46	l	
47		
48		
49	m	
4A	n	
4B		
4C	o	
4D		
4E		
4F	p	
50		
51	q	
52	r	
53		
54		
55		
56		
57	$	
58		
59		
5A		
5B		NL
5C		
5D		BS
5E		IL
5F		
60		
61	&	
62	a	
63		
64	b	
65		
66		
67	c	
68	d	
69		
6A		
6B	e	
6C		
6D	f	
6E	g	
6F		
70	h	
71		
72		
73	i	
74		
75		
76		Ⓨ
77		
78		
79		
7A		HT
7B		
7C		Dwnshft
7D		
7E		
7F		DEL

S/360 Byte (hex)	Graphic	Control
80		
81		SP
82	=	
83		
84	<	
85		
86		
87	;	
88	:	
89		
8A		
8B	%	
8C		
8D	'	
8E	>	
8F		
90	*	
91		
92		
93	(	
94		
95	)	
96	"	EOA
97		
98		
99		
9A		
9B		
9C		Upshift
9D		
9E		
9F		
A0	¢	
A1		
A2		
A3	?	
A4		
A5	S	
A6	T	
A7		
A8		
A9	U	
AA	V	
AB		
AC	W	
AD		
AE		
AF	X	
B0		
B1	Y	
B2	Z	
B3		
B4		
B5		
B6		
B7		Ⓢ
B8		
B9		
BA		
BB		Index Attn
BC		
BD		EOB
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1	-	Ⓝ
C2		
C3	J	
C4		
C5	K	
C6	L	
C7		
C8		
C9	M	
CA	N	
CB		
CC	O	
CD		
CE		
CF	P	
D0		
D1	Q	
D2	R	
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		NL
DC		
DD		BS
DE		IL
DF		
E0		
E1	+	
E2	A	
E3		
E4	B	
E5		
E6		
E7	C	
E8	D	
E9		
EA		
EB	E	
EC		
ED	F	
EE	G	
EF		
F0	H	
F1		
F2		
F3	I	
F4		
F5		
F6	¬	Ⓨ
F7		
F8		
F9		
FA		HT
FB		
FC		Dwnshft
FD		
FE		
FF		DEL

Figure 48. Line Code for IBM 2740 Communication Terminal

S/360 Byte (hex)	Graphic	Control
00		
01		SP
02	1	
03		
04	2	
05		
06		
07	3	
08	4	
09		
0A		
0B	5	
0C		
0D	6	
0E	7	
0F		
10	8	
11		
12		
13	9	
14		
15	0	
16	#	EOA
17		
18		
19		
1A		
1B		
1C		Upshift
1D		
1E		
1F		EOT
20	@	
21		
22		
23	/	
24		
25	s	
26	t	
27		
28		
29	u	
2A	v	
2B		
2C	w	
2D		
2E		
2F	x	
30		
31	y	
32	z	
33		
34		
35		
36		
37	,	(S)
38		BYP
39		
3A		
3B		Index Attn
3C		
3D		EOB
3E		PRE
3F		

S/360 Byte (hex)	Graphic	Control
40	-	(N)
41		
42		
43	j	
44		
45	k	
46	l	
47		
48		
49	m	
4A	n	
4B		
4C	o	
4D		
4E		
4F	p	
50		
51	q	
52	r	
53		
54		
55		
56		
57	$	
58		RES
59		
5A		
5B		LF-CR
5C		
5D		BS
5E		IL
5F		
60		
61	&	
62	a	
63		
64	b	
65		
66		
67	c	
68	d	
69		
6A		
6B	e	
6C		
6D	f	
6E	g	
6F		
70	h	
71		
72		
73	i	
74		
75		
76		(Y)
77		
78		
79		
7A		HT
7B		
7C		Downshift
7D		
7E		
7F		DEL

S/360 Byte (hex)	Graphic	Control
80		
81		SP
82	=	
83		
84	o	
85		
86		
87	;	
88	:	
89		
8A		
8B	%	(N)
8C		
8D	'	
8E	"	
8F		
90	*	
91		
92		
93	(	
94		
95	)	
96	±	EOA
97		
98		
99		
9A		
9B		
9C		Upshift
9D		
9E		
9F		
A0	¢	
A1		
A2		
A3	?	
A4		
A5	S	
A6	T	
A7		
A8		
A9	U	
AA	V	
AB		
AC	W	
AD		
AE		
AF	X	
B0		
B1	Y	
B2	Z	
B3		
B4		
B5		
B6		
B7	,	(S)
B8		
B9		
BA		
BB		Index Attn
BC		
BD		EOB
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0	-	(N)
C1		
C2		
C3	J	
C4		
C5	K	
C6	L	
C7		
C8		
C9	M	
CA	N	
CB		
CC	O	
CD		
CE		
CF	P	
D0		
D1	Q	
D2	R	
D3		
D4		
D5		
D6		
D7	!	
D8		RES
D9		
DA		
DB		LF-CR
DC		
DD		BS
DE		IL
DF		
E0		
E1	+	
E2	A	
E3		
E4	B	
E5		
E6		
E7	C	
E8	D	
E9		
EA		
EB	E	
EC		
ED	F	
EE	G	
EF		
F0	H	
F1		
F2		
F3	I	
F4		
F5		
F6	.	(Y)
F7		
F8		
F9		
FA		HT
FB		
FC		Downshift
FD		
FE		
FF		DEL

Figure 49. Hexadecimal Equivalents for IBM 2741 (BCD) Communication Terminal

S/360 Byte (hex)	Graphic	Control
00		
01		SP
02	1	
03		
04	2	
05		
06		
07	3	
08	4	
09		
0A		
0B	5	
0C		
0D	6	
0E	7	
0F		
10	8	
11		
12		
13	9	
14		
15	0	
16	#	EOA
17		
18		
19		
1A		RS
1B		
1C		UC
1D		
1E		
1F		EOT
20	@	
21		
22		
23	/	
24		
25	s	
26	t	
27		
28		
29	u	
2A	v	
2B		
2C	w	
2D		
2E		
2F	x	
30		
31	y	
32	z	
33		
34		
35		
36		
37	,	(S)
38		BY
39		
3A		
3B		LF
3C		
3D		EOB
3E		PRE
3F		

S/360 Byte (hex)	Graphic	Control
40		
41	–	(N)
42		
43	;	
44		
45	k	
46	l	
47		
48		
49	m	
4A	n	
4B		
4C	o	
4D		
4E		
4F	p	
50		
51	q	
52	r	
53		
54		
55		
56		
57	$	
58		RES
59		
5A		
5B		NL
5C		
5D		BS
5E		IL
5F		
60		
61	&	
62	a	
63		
64	b	
65		
66		
67	c	
68	d	
69		
6A		
6B	e	
6C		
6D	f	
6E	g	
6F		
70	h	
71		
72		
73	i	
74		
75		
76		(Y)
77		
78		
79		
7A		HT
7B		
7C		LC
7D		
7E		
7F		DEL

S/360 Byte (hex)	Graphic	Control
80		
81		SP
82	=	EOA
83		
84		
85	<	
86		
87	;	
88		
89	:	
8A		
8B	%	
8C		
8D	'	
8E	>	
8F		
90		
91	*	
92		
93	(	
94		
95	)	
96	"	
97		
98		
99		
9A		RS
9B		
9C		UC
9D		
9E		
9F		
A0	¢	
A1		
A2		
A3	?	
A4		
A5	S	
A6	T	
A7		
A8		
A9	U	
AA	V	
AB		
AC	W	
AD		
AE		
AF	X	
B0		
B1	Y	
B2	Z	
B3		
B4		
B5		
B6		
B7	!	(S)
B8		BY
B9		
BA		
BB		LF
BC		
BD		EOB
BE		PRE
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1	–	(N)
C2		
C3	J	
C4		
C5	K	
C6	L	
C7		
C8		
C9	M	
CA	N	
CB		
CC	O	
CD		
CE		
CF	P	
D0		
D1	Q	
D2	R	
D3		
D4		
D5		
D6		
D7	!	
D8		RES
D9		
DA		
DB		NL
DC		
DD		BS
DE		IL
DF		
E0		
E1	+	
E2	A	
E3		
E4	B	
E5		
E6		
E7	C	
E8	D	
E9		
EA		
EB	E	
EC		
ED	F	
EE	G	
EF		
F0	H	
F1		
F2		
F3	I	
F4		
F5		
F6	¬	(Y)
F7		
F8		
F9		
FA		HT
FB		
FC		LC
FD		
FE		
FF		DEL

Figure 50. Line Code (EBCD) for IBM 2741 Communication Terminal

S/360 Byte (hex)	Graphic	Control
00		
01		SP
02	1]	
03		
04	2	
05		
06		
07	3	
08	5	
09		
0A		
0B	7	
0C		
0D	6	
0E	8	
0F		
10	4	
11		
12		
13	0	
14		
15	z	
16	9	EOA
17		
18		
19		PN
1A		RS
1B		
1C		Upshft
1D		
1E		
1F		EOT
20	t	
21		
22		
23	x	
24		
25	n	
26	u	
27		
28		
29	e	
2A	d	
2B		
2C	k	
2D		
2E		
2F	c	
30		
31	l	
32	h	
33		
34		
35		
36		
37	b	
38		BYP
39		
3A		
3B		Index Attn
3C		
3D		EOB
3E		PRE
3F		

S/360 Byte (hex)	Graphic	Control
40	!	
41		
42		
43	m	
44		
45	.	
46	v	
47		
48		
49	'	
4A	r	
4B		
4C	i	
4D		
4E		
4F	a	
50		
51	o	
52	s	
53		
54		
55		
56		
57	w	
58		RES
59		
5A		
5B		LF-CR
5C		
5D		BS
5E		IL
5F		
60		
61	¡	
62	g	
63		
64	=	
65		
66		
67	f	
68	p	
69		
6A		
6B	;	
6C		
6D	q	
6E	,	
6F		
70	/	
71		
72		
73	y	
74		
75		
76	-	
77		
78		
79		
7A		Tab
7B		
7C		Dwnshft
7D		
7E		
7F		

S/360 Byte (hex)	Graphic	Control
80		
81		SP
82	± [	
83		
84	@	
85		
86		
87	#	
88	%	
89		
8A		
8B	&	
8C		
8D	ç	
8E	*	
8F		
90	$	
91		
92		
93	)	
94		
95	Z	
96	(	
97		
98		
99		
9A		
9B		
9C		Upshft
9D		
9E		
9F		EOT
A0	T	
A1		
A2		
A3	X	
A4		
A5	N	
A6	U	
A7		
A8		
A9	E	
AA	D	
AB		
AC	K	
AD		
AE		
AF	C	
B0		
B1	L	
B2		
B3	H	
B4		
B5		
B6		
B7	B	
B8		
B9		
BA		
BB		Index Attn
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0	°	
C1		
C2		
C3	M	
C4		
C5	.	
C6	V	
C7		
C8		
C9	"	
CA	R	
CB		
CC	I	
CD		
CE		
CF	A	
D0		
D1	O	
D2	S	
D3		
D4		
D5		
D6		
D7	W	
D8		
D9		
DA		
DB		LF-CR
DC		
DD		BS
DE		
DF		
E0		
E1	J	
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		Tab
FB		
FC		Dwnshft
FD		
FE		
FF		

Figure 51. Line Code (Correspondence) for IBM 2741 Communication Terminal

S/360 Byte (hex)	Graphic	Control
00		
01	T	
02		CR
03	O	
04		SP
05	H	
06	N	
07	M	
08		LF
09	L	
0A	R	
0B	G	
0C	I	
0D	P	
0E	C	
0F	V	
10	E	
11	Z	
12	D	
13	B	
14	S	
15	Y	
16	F	
17	X	
18	A	
19	W	
1A	J	
1B		FIGS
1C	U	
1D	Q	
1E	K	
1F		LTRS
20		
21	5	
22		CR
23	9	
24		SP
25	$\#_A$	$STOP_C$
26	$/_A$ $7/8_C$	
27		
28		LF
29	$)_A$ $3/4_C$	
2A	4	
2B	&	
2C	8	
2D	0	
2E	$:_A$ $1/8_C$	
2F	$;_A$ $3/8_C$	
30	3	
31	"	
32	$	
33	$?_A$ $5/8_C$	
34	$'_A$	$Bell_C$
35	6	
36	2_A $1/4_C$	
37	/	
38	-	
39	2	
3A		$Bell_A$
3B	$'_C$	FIGS
3C	7	
3D	1	
3E	$(_A$ $1/2_C$	
3F		LTRS

S/360 Byte (hex)	Graphic	Control
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
4A		
4B		
4C		
4D		
4E		
4F		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
5A		
5B		
5C		
5D		
5E		
5F		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
6A		
6B		
6C		
6D		
6E		
6F		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
7A		
7B		
7C		
7D		
7E		
7F		

S/360 Byte (hex)	Graphic	Control
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
CD		
CE		
CF		
D0		
D1		
D2		
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		
FB		
FC		
FD		
FE		
FF		

Figure 52. Line Code for AT & T 83B3 and WU 115A Terminals

S/360 Byte (hex)	Graphic	Control
00		NUL
01		NUL
02		
03	@	
04		SP
05		
06		
07		
08		
09		DLE
0A	P	
0B	P	
0C	0	
0D	0	
0E		
0F	p[1]	
10		
11		BS
12	H	
13	H	
14	(	
15	(	
16		
17	h	
18		CAN
19		CAN
1A		
1B	X	
1C		
1D	8	
1E	x[1]	
1E	x[1]	
20		
21		EOT
22	D	
23	D	
24	$	
25	$	
26		
27	d[1]	
28		Tp Aux Off
29		Tp Aux Off
2A		
2B	T	
2C		
2D	4	
2E	t	
2F	t	
30		FF
31		FF
32		
33	L	
34		
35		
36	'[2]	
37	'[2]	
38		
39		FS
3A	\	
3B	\	
3C	<	
3D	<	
3E		
3F	]	

S/360 Byte (hex)	Graphic	Control
40		
41		STX
42	B	
43	B	
44	"	
45	"	
46		
47	b[1]	
48		Tp Aux On
49		Tp Aux On
4A		
4B	R	
4C		
4D	2	
4E	r[1]	
4F	r[1]	
50		LF
51		LF
52		
53	J	
54		
55	*	
56	j[1]	
57	j[1]	
58		
59		SUB
5A	Z	
5B	Z	
5C	:	
5D	:	
5E		
5F	z[1]	
60		ACK
61		ACK
62		
63	F	
64		
65	&	
66		
67	f[1]	
68		
69		SYN
6A	V	
6B	V	
6C		
6D	6	
6E		
6F	v[1]	
70		
71		SO
72	N	
73	N	
74	•	
75	•	
76		
77	n[1]	
78		RS
79		RS
7A		
7B	↑	
7C		
7D	>	
7E	¬	
7F	¬	

S/360 Byte (hex)	Graphic	Control
80		SOH
81		
82	A	
83	A	
84	!	
85	!	
86	•	
87	a[1]	
88		X-On
89		X-On
8A		
8B	Q	
8C		
8D	1	
8E	q[1]	
8F	q[1]	
90		HT
91		HT
92		
93	I	
94		
95	)	
96	i[1]	
97	i[1]	
98		
99		EM
9A	Y	
9B	Y	
9C	9	
9D	9	
9E		
9F	y[1]	
A0		WRU
A1		WRU
A2		
A3	E	
A4		
A5	%	
A6	e[1]	
A7	e[1]	
A8		
A9		NAK
AA	U	
AB	U	
AC	5	
AD	5	
AE		
AF	u[1]	
B0		
B1		CR
B2	M	
B3	M	
B4	-	
B5	-	
B6		
B7	m[1]	
B8		GS
B9		GS
BA		
BB	]	
BC		
BD	=	
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		ETX
C1		ETX
C2		
C3	C	
C4		
C5	#	
C6	c[1]	
C7	c[1]	
C8		
C9		EOT X-Off
CA	S	
CB	S	
CC	3	
CD	3	
CE		
CF	s[1]	
D0		
D1		VT
D2	K	
D3	K	
D4	+	
D5	+	
D6		
D7	k[1]	
D8		ESC
D9		ESC
DA		
DB	[	
DC		
DD	;	
DE		
DF		
E0		
E1		Bell
E2	G	
E3	G	
E4	I	
E5	I g[1]	
E6		
E7	g[1]	
E8		ETB
E9		ETB
EA		
EB	W	
EC		
ED	7	
EE	w[1]	
EF	w[1]	
F0		SI
F1		SI
F2		
F3	O	
F4		
F5	/	
F6	o[1]	
F7	o[1]	
F8		
F9		US
FA	←	
FB	←	
FC	?	
FD	?	
FE		
FF		Rubout

Note 1: Lower case letters are converted to upper case in the terminal.

Note 2: Not all control characters are used by TWX but all are legitimate.

Figure 53. Line Codes for AT & T TWX Terminals

S/360 Byte (hex)	Graphic	Control
00		
01	T	
02		CR
03	O	
04		SP
05	H	
06	N	
07	M	
08		LF WRU
09	L	
0A	R	
0B	G	
0C	I	
0D	P	
0E	C	
0F	V	
10	E	
11	Z	
12	D	
13	B	
14	S	
15	Y	
16	F	
17	X	
18	A	
19	W	
1A	J	
1B		FIGS
1C	U	
1D	Q	
1E	K	
1F		LTRS
20		
21	5	
22		CR
23	9	
24		SP
25		
26	,	
27	.	
28		LF
29	)	
2A	4	
2B		
2C	8	
2D	0	
2E	:	
2F	=	
30	3	
31	+	
32		WRU
33	?	
34	'	
35	6	
36		
37	/	
38	-	
39	2	
3A		Bell
3B		FIGS
3C	7	
3D	1	
3E	(	
3F		LTRS

S/360 Byte (hex)	Graphic	Control
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
4A		
4B		
4C		
4D		
4E		
4F		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
5A		
5B		
5C		
5D		
5E		
5F		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
6A		
6B		
6C		
6D		
6E		
6F		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
7A		
7B		
7C		
7D		
7E		
7F		

S/360 Byte (hex)	Graphic	Control
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
CD		
CE		
CF		
D0		
D1		
D2		
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		
FB		
FC		
FD		
FE		
FF		

Figure 54. Line Code for IBM World Trade Telegraph ITA2

S/360 Byte (hex)	Graphic	Control
00		
01	T	
02		CR
03	O	
04		SP
05	H	
06	N	
07	M	
08		LF WRU
09	L	
0A	R	
0B	G	
0C	I	
0D	P	
0E	C	
0F	V	
10	E	
11	Z	
12	D	
13	B	
14	S	
15	Y	
16	F	
17	X	
18	A	
19	W	
1A	J	
1B		FIGS
1C	U	
1D	Q	
1E	K	
1F		LTRS
20		
21	.	
22		CR
23	:	
24		SP
25	?	
26	,	
27	7	
28		LF
29	)	
2A	/	
2B	0	
2C		Bell
2D	9	
2E	8	
2F	=	
30	-	
31		
32		WRU
33	6	
34	'	
35	5	
36	4	
37		
38	+	
39	3	
3A	2	
3B		FIGS
3C	1	
3D		
3E	(	
3F		LTRS

S/360 Byte (hex)	Graphic	Control
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
4A		
4B		
4C		
4D		
4E		
4F		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
5A		
5B		
5C		
5D		
5E		
5F		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
6A		
6B		
6C		
6D		
6E		
6F		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
7A		
7B		
7C		
7D		
7E		
7F		

S/360 Byte (hex)	Graphic	Control
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		

S/360 Byte (hex)	Graphic	Control
C0		
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
CD		
CE		
CF		
D0		
D1		
D2		
D3		
D4		
D5		
D6		
D7		
D8		
D9		
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3		
F4		
F5		
F6		
F7		
F8		
F9		
FA		
FB		
FC		
FD		
FE		DEL
FF		

Figure 55. Line Code for IBM World Trade Telegraph ZSC3

Appendix E: Running QTAM Application Programs Under TCAM

This section provides a brief summary of the changes needed to run a QTAM Message Processing Program using TCAM.

Using an Unmodified Existing Program

If the QTAM processing program is written so that the only QTAM macros issued are DCB, OPEN, CLOSE, GET, and PUT, the program need not be reassembled. Substitute the QTAM DD statements related to each process (input) and destination (output) DCB macro with corresponding TCAM DD statements. The format of the DD statement is

```
//ddname  DD  QNAME=procname
```

ddname is the symbolic name of the DD statement, and must be the same as the name specified in the DDNAME= operand of the process or destination DCB macro.

procname is the name of the process entry in the terminal table to which this entry refers. This name is assigned by the TPROCESS macro creating the entry. The destination queue may be changed at execution time by specifying a different value for the QNAME= parameter.

Reassembling a QTAM Processing Program

If macros other than OPEN, CLOSE, GET, PUT, and DCB are included in the application program, the program must be reassembled. A QSTART macro must be added as the first instruction of the program, immediately after the START or CSECT statement.

The QSTART macro distinguishes QTAM and TCAM application programs by indicating whether the QTAM program is to be assembled to run under QTAM or TCAM. QSTART is not coded in a TCAM application program, unless the CKREQ macro is used in a TCAM statement. There are no operands, and no assembler instructions are generated. The QSTART macro has an optional name field.

QTAM Macro Facilities

If a QTAM program is reassembled with a QSTART macro included, only some macro facilities are available. Password protection, as provided in TCAM, is not available. The following chart summarizes the macro facilities.

Macro	*Facility*
RETRIEVE	Transfers a message segment already placed on a destination queue or a process queue to a user-provided work area.
RELEASEM	Activates a designated terminal for receipt of message traffic from the CPU.
CLOSEMC	Initiates termination of the TCAM Message Control Program. Provides a flush closedown only.
STARTLN	Activates a designated line for operation.
STOPLN	Deactivates a designated line from operation.
COPYP	No Op.
COPYQ	No Op.
COPYT	No Op.
CHNGT	No Op.
CHNGP	No Op.

Appendix F: Summary of Operator Commands Classified by Operation

This appendix groups all commands according to the type of operation (for instance, DISPLAY, MODIFY, RELEASE) being performed. Figure 33 groups commands according to the areas in the TCAM system that are affected by issuing operator commands.

See *Operator Control* in the chapter *Using TCAM Service Facilities* for an explanation of operator command format and how to specify operator commands.

Operation	*Operands*	*Operator Command Name*
{DISPLAY / D}	TP,ACT,{grpname,rln / address}	ACTVATED
	TP,ADDR,statname	RLNSTATN
	TP,INACT,{grpname,rln / address}	INACTVTD
	TP,INTER	INTRCEPT
	TP,LINE,{grpname,rln / address}	LNSTATUS
	TP,LIST,{grpname,rln / address}	STATDISP
	TP,OPTION,statname,oplfdname	OPTFIELD
	TP,PRITERM	DPRIOPCL
	TP,QUEUE,statname	QSTATUS
	TP,SECTERM	DSECOPCL
	TP,TERM,statname	STSTATUS
{HALT / Z}	TP,{QUICK / FLUSH}	SYSCLOSE
{HOLD / H}	TP=statname	SUSPXMIT
{MODIFY / F}	id,AUTOPOLL={grpname,rln / address},OFF	AUTOSTOP
	id,AUTOPOLL={grpname,rln / address},ON	AUTOSTRT
	id,INTERVAL=POLL,statname,data	POLLDLAY
	id,INTERVAL=SYSTEM	INTERVAL
	id,INTERVAL=SYSTEM,data	SYSINTVL
	id,INTENSE=LINE,{grpname,rln / address},sense,count	ERRECORD
	id,INTENSE=TERM,statname,sense,count	ERRECORD
	id,OPERATOR={statname / SYSCON}	CPRIOPCL

Operation	*Operands*	*Operator Command Name*
	id,OPT=statname,opfldname,data	DATOPFLD
	id,TRACE={grpname,rln / address},OFF	NOTRACE
	id,TRACE={grpname,rln / address},ON	GOTRACE
{RELEASE / A}	TP=statname	RESMXMIT
{VARY / V}	statname,ONTP,B	ACTVBOTH
	statname,ONTP,E	ENTERING
	statname,OFFTP,B	NOTRAFIC
	statname,OFFTP,E	NOENTRNG
	{(grpname,rln) / grpname / address},OFFTP,{C / I}	STOPLINE
	{(grpname,rln) / grpname / address},ONTP	STARTLINE

Operator Commands Classified by Type of Operation.

Details presented in this section pertain to specific devices (except for the general information on binary synchronous (BSC) devices) supported by TCAM. Considerations are listed for start-stop and BSC devices; the final section of this appendix comprises considerations for the IBM 50 Magnetic Data Inscriber (MDI). More general information about the various subject matter contained here can be found in the appropriate chapters of this publication.

Start-Stop Devices

1030 Data Collection System

- On the STARTMH macro instruction, the CONV=YES operand must be coded if 1030 stations are included on lines processed by this message handler, since these stations do not have the capability of entering an EOT line control character after their messages. CONV=(opfield,switch) may be coded when the TERMINAL macro for the 1030 station initializes the option field to the specified setting.
- When the ADDR= operand of the TERMINAL macro is coded for a 1030 station, the two addressing characters must be immediately preceded by a 37 (this is the hexadecimal equivalent of the "circle S" character for the 1030. Neither the 37 nor the addressing characters are framed; the addressing characters must be the hexadecimal equivalent of the 1030 line code representation. Example: if the address of the 1030 is B1, the ADDR= operand of the TERMINAL macro would be coded as follows:

  ADDR=376402
  where 64 and 02 are the hexadecimal equivalent of the line-code representation of the characters B and 1, respectively.
- TCAM will recognize a message one data character in length from the IBM 1030.

1050 Data Communication System

- With regard to message translation, the character sets of the 1050 terminals contain lowercase as well as uppercase alphabetic characters. When messages from a 1050 are sent to stations or application programs that do not recognize codes for lowercase letters, the user should either enter only the uppercase form of alphabetic characters, or he should employ the 105F translation tables on the incoming side. These tables translate each incoming lowercase letter to the EBCDIC uppercase equivalent. These tables should also be specified if the source or destination of a message is entered at an IBM 1050 terminal in lowercase form; if the contents of the source or destination header field are not in uppercase form at the time an ORIGIN or FORWARD macro is executed, the header information is assumed to be invalid.
- The line correction feature is required if automatic retry is desired when a transmission error occurs.

1060 Data Communication System

- On the STARTMH macro instruction, the CONV=YES operand must be coded if 1060 stations are included on lines processed by this message handler, since these stations do not have the capability of entering an EOT line control character after their messages. CONV=(opfield,switch) may be coded when the TERMINAL macro for the 1060 station initializes the option field to the specified setting.

2260 Display Station (Remote)

- The last character of the invitation sequence (on the INVLIST macro instruction) for a remote 2260 must be X'40' (this is the hexadecimal representation of the Read MI character).
- When specifying the sequence of addressing characters for a remote 2260 on the TERMINAL macro instruction, the user must code one of the following control characters immediately after the addressing sequence:

  - X'A0' for a Write-DC operation;
  - X'B0' for a Write-at-Line-Address operation;
  - X'E0' for a Write-Erase operation.

  The three operations are described in the discussion of the SCREEN macro instruction.

- The 2260 translation table converts outgoing lowercase alphabetic characters to uppercase so that the terminal receives only uppercase characters.
- The MSGLIMIT macro instruction is recommended for use with this type of terminal; the outheader subgroup for 2260s should include a MSGLIMIT macro specifying a limit of one message in inquiry applications (in order to ensure that a response message is not erased before it can be read). Equal priority is recommended.
- The MSGFORM macro instruction and the LC=OUT operand of the STARTMH macro instruction should be used when sending to the 2260 from other devices.

2260 Display Station (Local)

NOTE: In coding the INVLIST, TERMINAL, and line group DCB macros for the 2260 Local configuration, you may consider each 2848 Control Unit attached locally to be a line group, and each 2260 station attached to such a control unit as a line in that group; that is, you may code one DD statement and one line group DCB macro per control unit and one INVLIST macro per terminal. In addition, each 2260 Local must be represented by a TERMINAL macro.

- Issue one INVLIST macro instruction for each IBM 2260 Local Display Station; this macro must contain a single entry for the station. All 2260 Locals attached to the same IBM 2848 Display Control Unit may be considered to be in the same line group, or each 2260 may be defined separately; the INVLIST macros for these stations must be specified in the INVLIST= operand of the line group DCB macro according to ascending relative line number. (Relative line number for 2260 Local stations in the same line group is determined by the order in which their TERMINAL macros are arranged; see the description of the TERMINAL macro.)

  The INVLIST entry for a 2260 Local should consist of the name of the station, a "+", and a one-byte code of X'02' (Read DS MI) indicating the type of Read operation to be performed when data is entered at the station. For further information about this Read command, see *IBM System/360 Component Description: IBM 2260 Display Station, IBM 2848 Display Control,* Order No. GA27-2700.

  Example:
  The following INVLIST macro is for a 2260 Local station named STA1.

```
LOCALST1  INVLIST  ORDER=(STA1+02)
```

  The X'02' causes a Read DS MI operation to be performed by TCAM when data is entered at the terminal.
- Issue one TERMINAL macro per 2260 Local. All TERMINAL macros for 2260s on the same 2848 Control Unit must be grouped together. Assign each terminal a relative line number according to the position of its TERMINAL macro in the group; i.e., in the first TERMINAL macro in the group, enter RLN=1, in the second, RLN=2, etc. The ADDR= operand is not meaningful for the 2260 Local.

  Example:
  The following TERMINAL macro is the first in a group of macros representing 2260 Local terminals attached to a 2848 Control Unit:

```
TERM1 TERMINAL QTY=L,RLN=1,DCB=DCB2260L,TERM=2260L,QUEUES=MO
```

- One line group DCB macro instruction may be coded for each locally attached 2848 Control Unit. The INVLIST= operand should be coded so that the order in which the INVLIST macros for the terminals attached to the control unit are named in the operand corresponds to the order in which the TERMINAL macros for the terminals attached to the control unit are arranged.

  Specify CPRI=S (or CPRI=E if a MSGLIMIT macro is used to limit the number of messages). Send priority for 2260 Locals is the same as that for nonswitched contention stations, described in the transmission priority section of the chapter *Defining Terminal and Line Control Areas.*

  If the user has keyed in part of a message he wishes to enter, but has not actually entered it at the time TCAM sends a message to his terminal, the message he is attempting to enter is erased from his screen and must be re-entered at a later time.

Example:
The following line group DCB macro is for an IBM 2848 Control Unit attached locally.

```
DCB2260L    DCB     DSORG=TX,MACRF=(G,P),          *
                    CPRI=S,DDNAME=DD2260L,         *
                    INVLIST=(LOCALST1,,,           *
                    LOCALST2,,,LOCALST3,,),        *
                    PCI=(N,N),BUFIN=1,BUFOUT=1,    *
                    BUFSIZE=400,BUFMAX=1
```

The following DD statement would be included in the job control cards for the execute steps, if the 2260s were assigned the addresses 150, 151, and 152:

```
//DD2260L   DD    UNIT=150
//          DD    UNIT=151
//          DD    UNIT=152
```

Dynamic PCI buffering is not recommended for the 2260 Local, as the data rate for this configuration is higher than for most other terminals.

- TCAM recognizes a one-character message entered by a 2260 local station (generally, a message must be at least two bytes long in order to be recognized by TCAM).

2265 Display Station

- The MSGLIMIT macro instruction should be used for this type of terminal; the out-header subgroup for 2265s should include a MSGLIMIT macro specifying a limit of one message in inquiry applications (in order to ensure that a response message is not erased before it can be read).
- Specifying receive priority with a user-determined delay may also be helpful.

2740 Communications Terminal

- With regard to message translation, the character sets of the IBM 2740 terminals contain lowercase as well as uppercase alphabetic characters. When messages from an IBM 2740 are sent to stations or application programs that do not recognize codes for lowercase letters, the user should either enter only the uppercase form of alphabetic characters, or he should employ the 274F translation tables on the incoming side. These tables translate each incoming lowercase letter to the EBCDIC uppercase equivalent. These tables should also be specified if the source or destination of a message is entered at a 2740 terminal in lowercase form; if the contents of the source or destination header field are not in uppercase form at the time an ORIGIN or FORWARD macro is executed, the header information is assumed to be invalid.

2740 Terminals with Station Control or Station Control and Checking:

- On the INVLIST macro instruction, the invitation sequence for this type of terminal consists of a single polling character, followed by a space character (X'01' in line code).
- When a TERMINAL macro is coded for a 2740 with these features, the addressing sequence consists of a single polling character. Immediately preceding this character, X'37' should be coded; immediately following the character, an X'01' should be coded.

2740 Terminals with Transmit Control or Transmit Control and Checking:

- When coding an INVLIST macro for a 2740 with these features, the following invitation sequence must always be specified: X'2301' (X and framing quotes are not coded).
- On the TERMINAL macro, no addressing sequence should be specified for a 2740 with these features.

2740 Basic Terminals:

- On the TERMINAL macro, no addressing characters should be coded for any of the four IBM 2740 Basic terminal configurations supported by TCAM.
- Send priority is the suggested method for using 2740 basic terminals.
- If equal priority is specified for a 2740 Basic terminal on a nonswitched line, messages may be entered at the terminal whenever the line is idle. The invitation list for this line may consist of one dummy entry (see the description of the INVLIST macro). The terminal operator may ask the computer to send by pressing the BID key and then

pressing the EOT key. The computer then sends all messages queued for the terminal. After all messages are sent, the computer is again ready to receive messages. Messages queued for the terminal will also be sent as soon as the terminal operator enters a number of consecutive messages in the sequence: BID key – message – EOT, which is equal to the number specified by a MSGLIMIT macro coded in the inheader subgroup of the message handler for this line (see the description of the MSGLIMIT macro).

2740 Basic Dial:

- TCAM uses a Prepare command on 2740 Basic terminals and there is no time-out constraint; consequently, an operator must enter BID EOT to indicate to TCAM that he has no message to enter.

2740 Terminals on a Switched Line:

- For 2740s on a switched line, after the terminal operator has finished entering his messages, he should press the BID key and then press the EOT key to indicate that he has no more messages to enter; otherwise, TCAM does not break the line connection and the terminal will eventually time out. *NOTE:* A 2740 Basic terminal on a switched line has a response command that does not time out.

2740 Model 2 Communication Terminal:

- The 2740-2 is defined as buffered by the BFDELAY= operand on the TERMINAL macro. (For more details, see the discussion of transmission priorities for nonswitched polled stations that use TCAM's buffering feature in the chapter *Defining Terminal and Line Control Areas.*)
- Send priority must be specified (see the description of the CPRI= operand of the line group DCB macro instruction).
- Queuing must be by terminal (see the description of the QBY=T operand of the TERMINAL macro instruction).
- Extended lock mode (obtained by the LOCK macro instruction) must not be used with this terminal because of the danger of tying up the line.
- The data portion of a message sent to an IBM 2740 Model 2 must not be longer than the length of the terminal's hardware buffer; otherwise, data in the buffer is overlaid and lost.

2741 Communications Terminal

- Can enter messages directed to other stations, but cannot receive messages from other stations. Can receive messages entered by itself.
- Can receive messages directed to it from an application program, but only if such messages are responses to inquiries from the terminal, and message lock or extended lock mode is specified by the LOCK macro.
- Messages directed to the 2741 terminal must not contain an EOT line-control character.
- Send priority must be specified by the CPRI= operand of the line group DCB macro.
- CALL=NONE must be specified in the TERMINAL macro for switched 2741 stations.
- For stations that do not perform parity and block checking (e.g., IBM 2740 Basic, IBM 2741, WTTC, TWX), you may wish to test for loss of incoming messages by coding a MSGGEN macro in your inmessage subgroup; this macro should test bit 25 (error during text transfer) of the message error record and send a message to the source indicating that the latest message entered has been lost and should be re-entered, if bit 25 is on.
- For 2741s on a switched line, after the terminal operator has finished entering all his messages, he should press the carrier-return key to indicate that he has no more messages to enter; otherwise, TCAM does not break the line connection.

2760 Optical Image Unit

- Line control for the 2760 is the same as for other 2740s and 1050s except for the advancement to the next frame on the screen.
- For the user to perform conversational operations only, he must code the LOCK macro and the CONV=YES operand on the STARTMH macro.
- The EOA sequence (X'16') is not written by TCAM; this character must be provided by the user or by the MSGFORM macro and must appear in the first position of each buffer.
- The user program also must provide the PRE O sequence (X'3E4C') that directs messages sent from the CPU to the 2760. If provided in an application program, X'27D6' is specified and the message is sent through an MH containing a CODE macro instruction.

7770 Audio Response Unit (ARU)

- Issue one INVLIST macro for each TCAM audio line; e.g., a line connected to an IBM 7770 Audio Response Unit, Model 3. This macro instruction assigns an invitation message to the line; the message is sent whenever a telephone or audio terminal calls in on the line. The operand of an INVLIST macro for an audio line has a single entry that consists of the name of the TERMINAL macro for the line over which the invitation message is to be sent, the active-/inactive-entry indicator, and an invitational message is specified as CPU ID.

  The vocabulary of the ARU resides on an analog drum; a track on the drum can contain one word of the ARU's vocabulary list. To specify his invitation message, the user codes a series of pairs of hexadecimal digits in the CPU ID entry: each pair represents the address of a track containing one word of the message. For example, in the entry

  BOS+09011B

  a message consisting of three words is specified. These words are located on tracks 09, 01, and 1B (hexadecimal notation) of the vocabulary drum.

  The name of the INVLIST macro for an audio line should be specified in the INVLIST= operand of the DCB macro for the line group containing the line.

  Example:
  The following INVLIST macro instruction creates the invitation list for an audio line.

Name	*Operation*	*Operand*
LIST10	INVLIST	ORDER=(BOS+09011B)

  BOS is the name of the TERMINAL macro instruction specifying UTERM=YES for this line. The + indicates that messages may be received on this line. The characters 09, 01, and 1B are the numbers (in hexadecimal notation) of the tracks on the ARU vocabulary drum that contain the words of the invitation message for this line.
- A TCAM audio line, i.e., a line connected to an IBM 7770 Audio Response Unit, Model 3, requires a TERMINAL macro instruction coded with the UTERM=YES operand.
- The LOCK macro instruction is required for audio terminals.

World Trade Telegraph (WTTA) Terminals

- For WTTA terminals, two message translation codes can be specified. These are International Telegraph Alphabet number 2 (ITA2) and Figure Protected Code (ZSC3).
- If equal transmission priority is specified for a WTTA terminal on a nonswitched contention line, messages may be entered at the terminal whenever the line is idle. Messages queued for the terminal will be sent only if a MSGLIMIT macro instruction is coded in the inheader subgroup of the Message Handler for this line, and then only when the number of messages entered by the terminal is equal to the number specified in the MSGLIMIT macro.
- TCAM does not support the WRU character on output.

Teletypewriter Exchange (TWX) Stations

- All entries in an invitation list must have the same number of invitation characters. If TWX stations on the same switched line are assigned ID sequences that differ in length, ID sequences shorter than the longest ID sequence specified in an INVLIST entry should be padded to the right with EBCDIC blanks to bring them up to the length of the longest sequence. The maximum-length TWX ID sequence supported by TCAM is 23 bytes (including framing control characters). It is recommended that each terminal ID sequence included as part of an invitation list entry be preceded and followed by certain control characters. These characters, and the hexadecimal representations of their line-code bit patterns (shown in non-parity TWX transmission code) are:

  a) characters

  CR LF idchars CR LF XON

b) hexadecimal representation

B151idcharsB15189

An entry for a TWX terminal named RAL that is assigned the ID sequence IBM 35ASR #1 might be coded:

RAL+B1519343B305CDAD83CB4B05C58DB15189

(If a TERMINAL macro coded UTERM=YES were issued for the line, the name of the TERMINAL macro would be coded in place of RAL.)

For lines to TWX terminals, it is recommended that the computer ID sequence also be preceded and followed by certain control characters. These characters, and the hexadecimal representations of their line-code bit patterns, are:

a) characters

Null CR LF Rubout idchars CR LF XON

b) hexadecimal representation

01B151FFidcharsB15189

If the ID sequence were RALEIGH, the operand for the computer ID might be coded:

CPUID=CPUNAME

Somewhere within the same area of the MCP the following field would be defined:

```
CPUNAME  DC  X'0E'
         DC  X'01B151FF4B8333A393E313B15189'
```

A table for translating TWX line code to hexadecimal representation is given in *Appendix D.*

Example:
The following INVLIST macro creates the invitation list for a switched line having three TWX terminals (named SCTN, PITT, and PHIL) assigned to it. Each of these terminals is assigned a unique ID sequence, consisting of its name. The computer is assigned the ID sequence PENN. It is assumed that the TWX terminals are non-parity machines.

Name	*Operation*	*Operands*
LIST1	INVLIST	(SCTN+B151CBC32B73B15189, * PITT+B1510B932B2BB15189, * PHIL+B151B131933B15189), * CPUID=TWXADDR

Here, CBC32B73, 0B932B2B, and 0B131933 are the TWX non-parity transmission-code representations of the ID sequences SCTN, PITT, and PHIL, respectively, in hexadecimal notation. B1, 51, and 89 are the non-parity hexadecimal representations of the TWX CR, LF, and XON line-control characters, respectively. Somewhere in the MCP the following field is defined:

Name	*Operation*	*Operand*
TWXADDR	DC DC	X'0B' X'01B151FF0BA37373B15189'

In this instance, 0B is the hexadecimal length of the rest of the field. 0BA37373 is the non-parity TWX transmission-code representation of the ID sequence PENN, in hexadecimal notation. 01, B1, 51, FF, and 89 are the non-parity hexadecimal representations of the TWX Null, CE, LF, Rubout, and XON line-control characters, respectively.

- Transmission priority for TWX stations is as follows: the computer invites the station to enter a message by sending the computer ID sequence to the station. The computer ID sequence is sent after each message is entered by the station to invite the station to enter another message. When the station has no more messages to enter, the station operator should so inform the computer by pressing the XOFF key after receiving the computer ID sequence.
- Two types of TWX terminals may be used with TCAM. The first of these enters and accepts parity data. For this type of TWX station, the TTYB translation table is provided. The second type of TWX station enters and accepts only non-parity data; i.e., the parity bit must be one in all characters. The TTYC translation table is provided for translating data received and sent to TWX terminals of this type. The user may wish to receive messages from or send messages to both type of TWX terminals over lines in the same line group, in which case he may issue two CODE macros in his incoming or outgoing group and route each message to one or the other, as described in the section *Variable Processing within a Message Handler.*
- The user should end all TWX messages with the XOFF control character instead of with the EOT line control character. If EOT is used, the line is disconnected and a console error message is posted.

AT&T 83B3 Selective Calling Station

- When specifying the sequence of addressing characters on the TERMINAL macro instruction for an AT&T 83B3 terminal, the user must code a LTRS (letters-shift) control character immediately after the two addressing characters. The LTRS character is specified by coding X'1F'. The X and framing quotes are not coded.

Binary Synchronous Communication (BSC) Terminals

In this section, information that is applicable to BSC devices in general will be presented first; following this general information will be sections dealing with each BSC station supported by TCAM.

TCAM Send and Receive Operations on a BSC Line

This section presents TCAM's responses to various line-control characters entered by a station during an attempt on the part of TCAM to read messages entered by that station or to send messages to it, and describes conditions that TCAM interprets as errors during invitation, selection, and transmission. This information will be of interest to those who are programming a computer to serve as a remote station in a TCAM system, to the user whose application demands a knowledge of TCAM's line-control scheme, and to those interested in the various conditions that prompt TCAM to set bits in the message error record having to do with errors encountered while TCAM is reading or writing text on a BSC line. For general information on BSC line control, see the publication *General Information-Binary Synchronous Communications* (Order No. GA27-3004). For detailed information on TCAM's channel programs and error-recovery procedures, see the *TCAM Program Logic Manual.*

In the following two sections, the statement is made in many places that TCAM's response to a particular error is to terminate its attempt to send or receive the current message. In this case, TCAM attempts to send or to receive the next eligible message. If no provision is made in the outmessage subgroup of an MH to test whatever error bits were set and to take appropriate action following an error, the message is treated by TCAM as if it had been successfully transmitted; no further attempt is made to send it. On the receiving side, if any data in the message in error is received, the portion of the message that was received is sent through the incoming group of the appropriate MH and is placed on the destination queue for the destination (if the destination is known).

Sending Operations

- If TCAM sends out an ENQ or an addressing sequence and receives a response other than ACK, WACK, or (on a multipoint line only) RVI, TCAM attempts six more times to elicit a satisfactory response; if none is received, TCAM sets bit 24 (selection error) in the message error record and terminates its attempt to send the current message.
- If TCAM sends out an ENQ or an addressing sequence and receives an EOT in response, TCAM immediately sets bit 24 (selection error) in the message error record and terminates its attempt to send the current message.

For all point to point stations (including switched stations):

- If TCAM receives a WACK in response to an ENQ (indicating that the station's hardware buffer is busy), TCAM responds by writing another ENQ; this exchange is repeated until TCAM receives a response other than WACK. (A WACK in response to an ENQ can be received from an IBM 2770 station and from CPUs used as stations).
- If TCAM receives a NAK in response to an ENQ, TCAM sets bit 24 (selection error) in the message error record and sends an EOT to the station, thereby terminating its attempt to send the current message.
- For point-to-point nonswitched contention stations, if TCAM receives an ENQ in response to an ENQ (indicating a contention situation) TCAM always yields; that is, TCAM automatically delays the current attempt to send, giving the station an opportunity to enter data. In this case, TCAM attempts to send the message later.

For multipoint stations:

- If TCAM receives a NAK in response to an addressing sequence, TCAM sets bit 24 (selection error) in the message error record and terminates its attempt to send the current message.
- If TCAM receives an RVI in response to an addressing sequence and TCAM's buffered terminal support is specified (by the BFDELAY= operand of the TERMINAL macros for stations on the line), TCAM re-addresses the station at a later time, if the addressing characters were for the first segment in the message; if they were for a segment other than the first, TCAM sets bit 25 (error during text transfer) and bit 7 (cutoff/RVI error) in the message error record and terminates its attempt to send the current message. If RVI is received in response to addressing characters for a station that is not using TCAM's buffered terminal support, TCAM re-sends the addressing sequence at a later time.

For all BSC stations:

- If TCAM receives, as a response to a block of text, a character other than ACK, NAK, WACK, RVI, or EOT, or receives no response at all, TCAM sends the station an ENQ six times; if none of the six ENQs elicit a satisfactory response, TCAM sets bit 25 (error during text transfer) in the message error record and sends an EOT to the station, thereby terminating its attempt to send the current message.
- If TCAM receives an EOT in response to text, TCAM immediately sets bit 14 (BSC abort) and bit 25 (error during text transfer) in the message error record and terminates its attempt to send the current message.
- If TCAM receives a NAK in esponse to text, TCAM re-sends the block of text up to six times; if no satisfactory response is elicited, after six retries TCAM sets bit 25 (error during text transfer) in the message error record and sends the station an EOT, thereby terminating its attempt to send the current message.
- If TCAM receives an RVI in response to text, TCAM considers this to be a normal response and continues sending. If an RVI is received as the response to two successive blocks of text, TCAM sends an ENQ in response to the second RVI. If another RVI is received in esponse to the ENQ, TCAM sends another ENQ; after six ENQ/RVI exchanges, TCAM sends the station an EOT and terminates its attempt to send the current message.
- If TCAM receives a WACK in response to text sent to a BSC station using TCAM's buffered terminal support (indicating that the terminal's buffer space is exhausted), TCAM sends the station an EOT and sends the next block of text after the interval specified by the BFDELAY= operand of the TERMINAL macro has been observed. If WACK is received in response to a block of text sent to a station not using TCAM's buffered terminal support, TCAM sends an ENQ; if another WACK is received, the cycle is repeated until a different response is received by TCAM.

Receiving Operations

- Each block of incoming data must begin with a valid start character (SOH, STX, DLE/STX) and end with a valid end character (ETB, ETX, EOT). (In order to be valid the EOT character must be transmitted by itself, as a separate block.) When this requirement is not met, TCAM assumes that an error has occurred and requests retransmission of the block. If the error has not been corrected after six retries, TCAM

sets bit 21 (format error) and bit 25 (error during text transfer) in the message error record, and sends the station an EOT character.

- Each incoming block of a message is required to be in the same transmission mode (transparent or nontransparent) as the other blocks of the message. If all blocks of the same message are not in the same transmission mode, TCAM sets bit 21 (format error) and bit 25 (error during text transfer) in the message error record.
- When TCAM receives an ENQ character while attempting to read, TCAM sends back the previous acknowledgment character (ACK0 or ACK1).
- When TCAM receives a TTD (temporary text delay, STX-ENQ sequence, TCAM responds with a NAK; this is not considered to be an error condition. When TCAM receives a TTD and the next block read in is an ET (indicates a possible station malfunction resulting in a truncated message for a 2770 or 2780 station), TCAM sets bit 14 (BSC abort) and bit 25 (error during text transfer) in the message error record and terminates its attempt to send the current message.
- When TCAM receives a station ID sequence on a switched BSC line, TCAM compares invitation lists for a matching ID sequence, beginning with the invitation list for the line over which the ID sequence was received and working upward through all lines in the line group having higher relative line numbers than the line over which the ID sequence was received.
- If no match is found for a station ID sequence read in over a switched line, TCAM attempts to read in the ID sequence and find a match for it six more times. If these attempts are unsuccessful, TCAM sets bit 17 (invalid station ID) in the message error record, and terminates its attempt to send or receive the message. In addition, TCAM breaks the line connection with the station.
- If TCAM dials a BSC station on a switched line and receives an invalid ID sequence, TCAM re-sends the computer's ID-ENQ sequence six times; if no valid ID sequence is received from the station after six retries, TCAM sets bit 17 (invalid station ID) in the message error record and breaks the telephone connection, thereby terminating its attempt to send the message eligible to be sent first to the station.

Other BSC Considerations

- EOB checking must be specified by the STARTMH macro if the Message Handler is to handle messages whose origin or destination is a BSC station (for directions on specifying EOB checking, see the description of STARTMH).
- On the INVLIST macro instruction, all entries in an invitation list must have the same number of invitation characters. This requirement presents a problem with respect to BSC stations on a multipoint line. BSC stations are compatible; that is, more than one type of BSC station may be included on the same line. Since different types of polled BSC stations require different numbers of polling characters (e.g., a polled IBM 1130 requires one polling character, while a polled IBM 2780 requires two), some polling sequences in an invitation list for a line connecting different kinds of BSC terminals may have to be padded to bring them up to the length of the longest sequence. Synchronous idle characters are used to bring each sequence of polling characters specified in an INVLIST entry for a polled BSC station up to the length of the longest sequence specified in any entry for that INVLIST macro. These characters are inserted to the left of the polling characters. The synchronous idle characters used depend upon the transmission code for the station: appropriate characters are

  - for EBCDIC, X'32';
  - for ASCII, X'16';
  - for 6-bit Transcode, X'3A'.

  The X and the framing quotes are not coded.

  When creating an INVLIST entry for a polled BSC station, code the ENQ line control character after each sequence of hardware polling characters. Appropriate characters are

  - for EBCDIC, X'2D';
  - for ASCII, X'05';
  - for 6-bit Transcode, X'2D'.

  The X and the framing quotes are not coded. For example, the entry for a polled IBM 2780 terminal named NYC that uses EBCDIC transmission code might be coded

  ```
  NYC+C1F62D
  ```

where C1F6 is the EBCDIC representation of the polling characters A6 in hexadecimal notation, and 2D is the EBCDIC representation of the ENQ line control character in hexadecimal notation.

- The EOT= operand of the INVLIST macro must be coded for BSC stations on a multipoint line.
- TCAM supports four BSC ID schemes; the user can specify a CPU ID and no station ID, a station ID and no CPU ID, both a station ID and a CPU ID, or neither a station ID nor a CPU ID.
- When BSC stations are assigned ID sequences, the unit size (as specified in the KEYLEN= operand of the INTRO macro) must be at least as long as the sum of the longest ID sequence that can be entered by a station plus one byte; otherwise, errors may occur.
- If a unique ID is assigned each BSC station on a switched line, no origin macro is needed to identify these stations.
- The length of a CPU or station ID sequence should not be longer than eight characters.
- If no ID exchange is desired for BSC stations on a switched line, the invitation list for the line should consist of a single entry containing the name of a single station on that line followed by a +. Example: TERMA+
- If a one-character station ID sequence is being used, the EBCDIC pad character (X'DF') should not be assigned as an ID.
- If BSC stations on the same switched line are assigned ID sequences that differ in length, ID sequences shorter than the longest ID sequence specified in an INVLIST entry should be padded to the right with EBCDIC blanks to bring them up to the length of the longest sequence. The maximum length BSC ID sequence is 15 bytes.
- The sequence of addressing characters for a BSC station must be followed by the ENQ line control character. Appropriate characters are:

  - for EBCDIC, X'2D';
  - for ASCII, X'05';
  - for 6-bit Transcode, X'2D'.

  The X and framing quotes are not coded.
- A switched line to BSC stations that are all assigned unique ID sequences does not require a TERMINAL macro instruction coded UTERM=YES. For such a line, the user should enter each station's name and ID sequence and the CPU ID sequence in the appropriate operands of the INVLIST macro instruction for the line.
- If equal priority is specified for BSC point-to-point stations on a nonswitched contention line, messages may be entered at the terminal whenever the line is idle. Messages queued for the terminal will be sent only if a MSGLIMIT macro instruction is coded in the inheader subgroup of the message handler for this line, and then only when the number of messages entered by the terminal is equal to the number specified in the MSGLIMIT macro instruction.
- Transmission priority for switched stations: When a BSC station calls the computer, the computer allows the station to enter one message after the connection is established. The BSC station enters a message if it has one. The computer then sends the BSC station one message, if any is queued for the station, and if the calling station has identified itself by means of an ID sequence or an origin field, verified by an ORIGIN macro, in the message header. Messages are sent by the computer according to the priority scheme outlined in *Message Priority and Queuing.* The computer alternates between sending the BSC station a single message and pausing to permit the station to enter a single message until the message queue for the station is exhausted. If no input message is entered immediately, the computer pauses for nine seconds before sending the station the next message on its destination queue. (A computer serving as a remote station can avoid the nine-second delay by entering an EOT when it has no message to enter.) When the last incoming message is received and no further messages appear on the destination queue for the station, the computer breaks the line connection, making the line available for new calls.

  This scheme also applies when the computer calls the station, except that the computer sends the first message (if it has one for the station) in this case. If the computer has nothing to send the station, and nothing comes in from the station, the computer breaks the line connection.

  If a BSC station calls the computer and does not identify itself (that is, does not use ID verification), the computer allows the station to enter messages and sends the station messages queued for this line. If no messages are queued for the line entry, and

if the station has not identified itself by the time it enters its last message, the computer breaks the line connection after giving the station a chance to enter and after receiving no message from the station.

IBM 2770 Data Communications System

- If 2770s are on multipoint lines, it is recommended that the stations be defined as buffered (by the BFDELAY= operand on the TERMINAL macro). Enough delay time should be specified by BFDELAY= to allow the terminal to empty one of its two hardware buffers. (TCAM's buffered terminal support is designed to optimize line usage for the 2770.)
- Only multipoint 2770 stations should be defined as buffered.
- Send priority must be specified for buffered terminals (see the description of the CPRI= operand of the line group DCB macro instruction).
- Queuing must be by terminal (see the description of the QBY= operand of the TERMINAL macro instruction).
- If the CONV= operand of the STARTMH macro specifies conversational mode for an MH handling messages being sent to a 2770 station, and if the station is set up so that selection of an output device is required, a device-selection character (X'11', '12', or '13', depending upon the output device selected) must be specified as the first character following the STX line-control character in each message sent to the 2770 station. The user must see to it that this character appears in his outgoing message—TCAM does not support transparent mode for messages being sent to 2770 stations requiring selection of an output device.
- When attempting to select a point-to-point 2770 or 2780 station, TCAM sends an ENQ character, the station responds with ACK0, and TCAM writes the escape sequence for the desired station component. If an RVI is received from the station instead of ACK0, TCAM sends an EOT to the station and attempts to send the message at a later time.
- Extended lock mode (obtained by the LOCK macro) must not be used with this terminal if TCAM's buffered terminal support is specified for it (by the BFDELAY= operand of the terminal macro).

IBM 2780 Data Transmission Terminal

- On the STARTMH macro instruction, the CONV=YES operand should *not* be coded if any 2780s are included on lines handled by this Message Handler; if it is coded, the BSC line integrity is destroyed.
- For a 2780 point-to-point, when the computer sends information to the punch or printer, an escape sequence must be specified in the TERMINAL macro (ADDR=escape sequence).
- When attempting to select a point-to-point 2770 or 2780 station, TCAM sends an ENQ character, the station responds with ACK0, and TCAM writes the escape sequence for the desired station component. If an RVI is received from the station instead of ACK0, TCAM sends an EOT to the station and attempts to send the message at a later time.

The TPEDIT Macro Instruction for the IBM 50 Magnetic Data Inscriber

The IBM 50 Magnetic Data Inscriber (MDI) is a key-operated device that records data on cartridge-contained magnetic tape. (For a description of the IBM 50 MDI, see the publication *IBM 50 Magnetic Data Inscriber Component Description,* Order No. GA27-2725.) It enables the user to enter data from source documents to magnetic tape, which is then used to enter data into an IBM System/360 through the IBM 2772 Control Unit. Data received from the IBM 50 MDI attachment to the IBM 2772 Multi-Purpose Control Unit contains MDI control characters; the TPEDIT macro allows the user to edit this data.

The re-enterable editing routine is activated by the TPEDIT macro issued in an application program following a GET or a READ macro (it must not be issued in the MCP); it edits MDI control characters as specified by operands in the TPEDIT macro. One of the operands also specifies whether a user error-exit routine is to handle error records. If data is to be received from more than one IBM 50 MDI at a time in an application program, a separate parameter list must be issued for each (described in more detail in *Input to the TPEDIT Macro*) below.

TPEDIT Macro Format

Name	*Operation*	*Operand*
[name]	TPEDIT	MINLN=n ,EDIT={EDITR / <u>EDITD</u>} ,RECFM={U / <u>V</u>} ,ERROPT={name / <u>IGNORE</u>} ,VERCHK={VOKCHK / <u>NOCHK</u>} ,REPLACE={X'xx' / <u>X'19'</u>} ,BUFFER={YES / <u>NO</u>}

name

Is the name of the macro and is optional. If included, the name may be any symbol valid in the assembler language.

MINLN=n

Specifies the minimum acceptable length of an input record where *n* is replaced by the decimal number of bytes desired as a minimum. For EDIT=EDITD, SOR and EOR codes are excluded from the length; for EDIT=EDITR, SOR and EOR are included in the length. This operand may not be omitted.

EDIT={EDITR / <u>EDITD</u>}

Specifies the type of editing to be done.

EDITR causes the input to be edited and the start-of-record (SOR) and end-of-record (EOR) delimiters to be retained as part of the output.

EDITD causes the input to be edited and start-of-record and end-of-record delimiters to be deleted. EDITD is assumed if this operand is omitted.

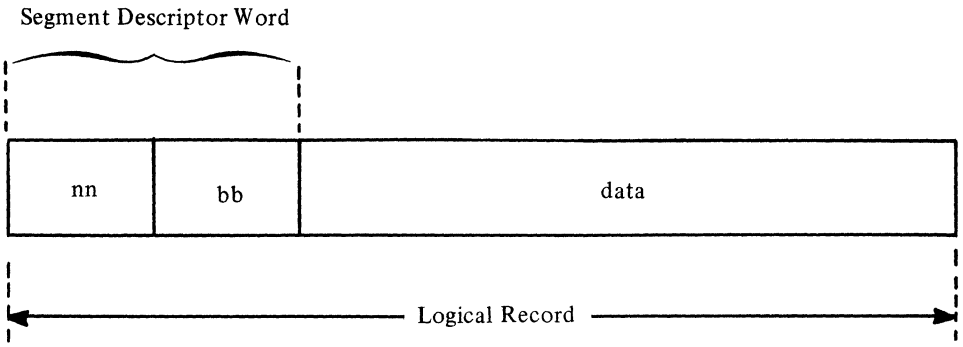
The edit consists of the following functions: records are extracted one at a time from the input area by scanning for the record delimiting codes (SOR and EOR). Duplicating (DUP) codes are replaced by the character(s) from the corresponding location of the record in the work area when control was last returned to the editing routine (true for all records except the first). Left-zero fields are right-adjusted, with leading zeros inserted where necessary. Left-zero start codes, group separator codes, and records containing a cancel code do not appear in the output stream. Line-control characters (ETB, ETX, STX, and DLE,STX) are always deleted if found in the input area.

RECFM={U / <u>V</u>}

Specifies the format of the output from the editing routine.

If RECFM=U is coded, no segment descriptor word is added to each record.

If RECFM=V is coded, a segment descriptor word is added to each record as shown (RECFM=V is assumed if this operand is not coded).



where *nn* (2 bytes) is the length of the logical record and *bb* (2 bytes) is binary zeros reserved for system use.

This four-byte field is included in the record length returned to the user in a parameter list.

NOTE: This four-byte field must be allowed for by the user when determining the size of the work area (see *Input to the TPEDIT Macro*) below.

ERROPT={name / IGNORE}

Specifies whether a user error exit routine is provided to handle error records.

name specifies the name of the user error exit routine to be entered when the editing routine detects logical errors or replacement characters in the record.

IGNORE specifies that an error exit routine is not provided. The error conditions are to be disregarded and the record is to be passed normally to the user. If this operand is omitted, ERROPT=IGNORE is assumed.

VERCHK={VOKCHK / NOCHK}
(valid only if ERROPT=name is coded)

Specifies whether the records are to be checked for verify OK (VOK) codes. NOCHK is assumed if this operand is omitted (and subsequent records are not checked).

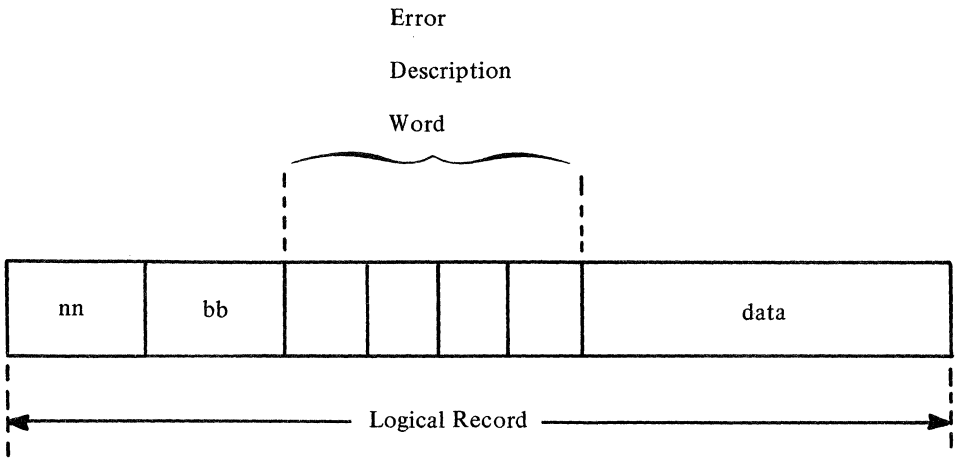
If VOKCHK is specified and a record does not contain the verify OK code, the record is passed to the error exit routine.

When the editing routine encounters an erroneous record and control passes to this user-supplied routine, register 13 contains the address of a 72-byte register save area aligned on a fullword boundary, and register 1 contains the address of a two-word parameter list aligned on a fullword boundary. The parameter list is defined as follows:

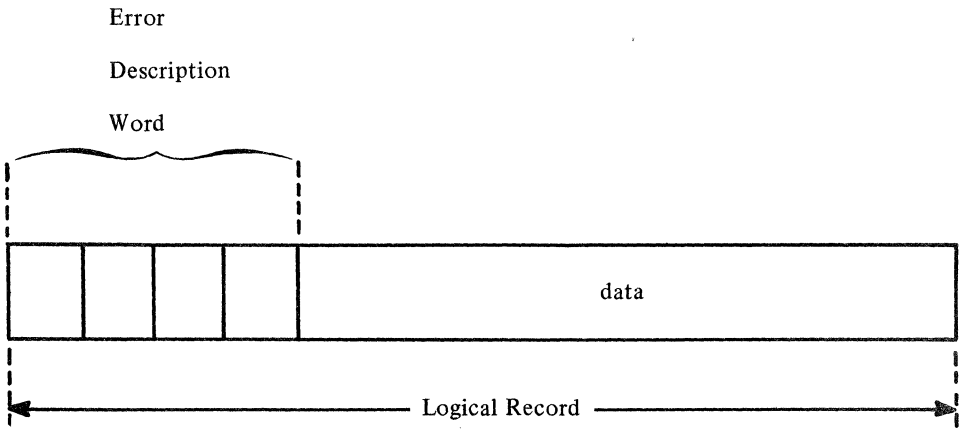
Word	*Contents*
1	Record address
2	Address of record length

The record length includes the four-byte Error Description Word appended, as shown, to the data record. In addition, if RECFM=V is coded in the TPEDIT macro, the logical record length*(nn)* includes these four bytes when it is passed to the error exit routine.

If RECFM=V is specified:



If RECFM=U is specified:



The contents of the Error Description Word are shown below. Further information is contained in a later section, *Identifying Records in Error.*

The error exit routine can be used to analyze and possibly correct a record that is in error. When control returns to the editing routine with a BR 14 instruction, the user must set register 15 to zero if the erroneous record is to be bypassed. Register 15 must be set to a nonzero value in order to direct the editing routine to ignore an error in the erroneous record, and thereby process the record in the normal manner. Whether the record in error is accepted or bypassed does not change its effect on subsequent records. The Error Description Word is removed by the editing routine when control returns from the error exit routine.

REPLACE={X'xx'} / {X'19'}

Specifies the code to be used as a replacement character when the editing routine detects a 2772 replacement character (EBCDIC SUB character, X'3F') in the input. X'19' is the assumed value (default) because it is an end-of-data (ED) signal for an IBM 50 MDI cartridge and therefore can never appear as a valid data byte.

For REPLACE=X'xx', the user can replace *xx* with any hexadecimal characters he chooses (note that *xx* must be enclosed in single quotes). Choices may be made from the code chart in Figure 56 with exceptions as noted below.

Programming note: BSC control characters should not be used as replacement characters if the data is to be transmitted by BSC facilities after editing.

Hexadecimal characters representing special purpose MDI codes that should *not* be used as replacement bytes are:

X'00'	(LZ)	X'1E'	(VOK)	X'74'	(P4)
X'11'	(DUP)	X'3C'	(RM)	X'75'	(P5)
X'12'	(LZS)	X'71'	(P1)	X'76'	(P6)
X'18'	(CAN)	X'72'	(P2)	X'77'	(P7)
X'1D'	(GS)	X'73'	(P3)	X'78'	(P8)

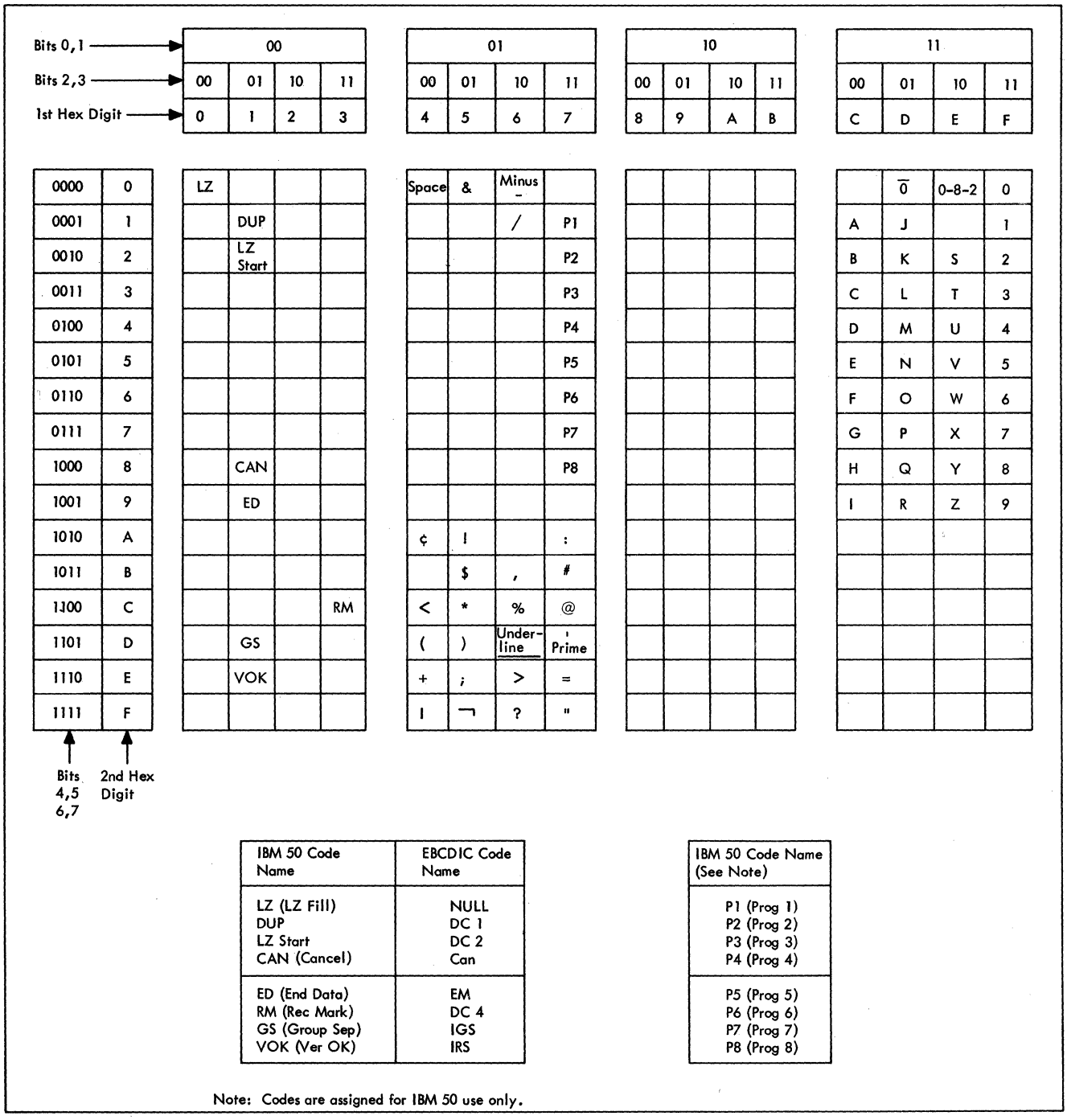
BUFFER={YES} / {NO}

This operand specifies whether the user's data is in BTAM buffers. TCAM users should either omit this operand or code BUFFER=NO.

Input to the TPEDIT Macro

Register 1 must point to a four-word parameter list (aligned on a fullword boundary) containing:

Word	*Contents*
1	Input Address This is the address of the data to be edited.
2	Input Length This is the length of the data to be edited.
3	Edit work area address The work area required by the editing routine for a given parameter list is obtained in either of two ways. The work area can be provided by the editing routine (by an unconditional GETMAIN), or it can be provided by the user. If the work area is to be provided by the editing routine, this word must contain binary zeros. The editing routine issues a GETMAIN macro to obtain the required storage, and places the address of the storage obtained in this word. If the work area is provided by the user, then this word contains the address of the area supplied. The amount of storage needed in addition to the fixed amount required is determined from: a) the maximum record length; b) whether a user exists (72 bytes for a register save area and four bytes for an EDW are required by the macro if an exit is specified); c) whether variable record formats are used; The size (in bytes) of the work area may be determined from the formula: S= 84 + 76E + R + 4V where S is the size (in byes of the work area E= 0 if ERROPT=IGNORE is coded E= 1 if ERROPT=name is coded V= 0 if RECFM=U V= 1 if RECFM=V R is the length of the longest record to be processed



IBM 50 Code Name	EBCDIC Code Name
LZ (LZ Fill)	NULL
DUP	DC 1
LZ Start	DC 2
CAN (Cancel)	Can
ED (End Data)	EM
RM (Rec Mark)	DC 4
GS (Group Sep)	IGS
VOK (Ver OK)	IRS

IBM 50 Code Name (See Note)
P1 (Prog 1)
P2 (Prog 2)
P3 (Prog 3)
P4 (Prog 4)
P5 (Prog 5)
P6 (Prog 6)
P7 (Prog 7)
P8 (Prog 8)

Note: Codes are assigned for IBM 50 use only.

Figure 56. IBM 50 MDI Control Codes

4 Maximum record length
This is the length, in bytes, of the longest valid edited record. For EDIT=EDITD the length should *exclude* SOB and EOB codes; for EDIT=EDITR, the length should *include* SOR and ER codes.

The value of the maximum record size should not include the four-byte Segment Descriptor Word added to a variable length record.

Records that exceed the maximum record size are considered error records.

Register 13 must contain the address of a 72-byte register save area aligned on a fullword boundary.

Return Codes

After the editing routine has edited a record, it provides a return code in register 15 indicating record availability and status of the input area, prior to returning control to the user. The return codes and their meanings are:

Code (hex)	*Meaning*
00	A record is available; input area is empty. The routine has edited the last logical record in the input area and is passing the record to the user.
04	A record is available; input area is not empty. The routine has edited one logical record and is passing that record to the user.
08	No record is available; input area is empty. The last record in the input area was incomplete; i.e., it was a partial record.
0C	End-of-cartridge (ED) code was detected.

For return codes 00 and 04, the record address and the address of the record length are given to the user in a two-word parameter list aligned on a fullword boundary. The address of the parameter list is returned in register 1. The parameter list has the following format:

Word	*Contents*
1	Address of the record
2	Address of the record length

Identifying Records In Error

This section describes what the editing routine considers to be records in error. Once a record is determined to be in error, the editing routine passes the record to the user error exit routine, if ERROPT=name is specified in the TPEDIT macro statement. If an error exit routine is not specified, the erroneous record is returned normally to the user.

The editing routine maintains information about each record as it is being edited. This information is summarized in the error description word (EDW) described below. When the EDW contains a non-zero value in either the level status (byte 0) or the type status (byte 1), the record is considered an erroneous record, and the EDW is inserted between the four-byte record length field and the data portion if RECFM=V is specified in the TPEDIT macro. Otherwise, the EDW is added to the start of the record to aid the use in analyzing the error.

Level Status (Byte 0)

The level status indicator identifies erroneous records that result from inter-record dependency and that cannot be identified in the type status byte.

The level status is presented with each error record and has a value of:

Format of Error Description Word for the TPEDIT Macro.

0 For any error record that will not cause questionable data in following record(s).

1 For any error record that may cause questionable data in following record(s), and the level status of the previous record was zero.

2 For any error record that has questionable data because of the error level of preceding record(s) that may cause questionable data in the following record(s), and where the level status of the previous record was either 1 or 2.

Byte 0 *Level status*

0	For any error record that will not cause questionable data to be in the following record(s).
1	For any error record that may cause questionable data to be in the following record(s).
2	For any error record that (1) contains questionable data due to the error level of preceding record(s) and (2) may cause questionable data to be in the following record(s); and where the level status of the previous record was either 1 or 2.

Byte 1 *Type status*

0	No identifiable error(s)
1	Start-of-record (SOR) or end-of-record (EOR) in error
2	Length error
4	Field error
8	Data check error

NOTE: This field may contain combinations of these error types; e.g., a C (hexadecimal) indicates a data check error *and* a field error.

Byte 2 *Program Level*

1 – P1	5 – P5	E – none of the preceding levels.
2 – P2	6 – P6	Start-of-record (SOR) is in error.
3 – P3	7 – P7	
4 – P4	8 – P8	

Byte 3 *Record Status*

U	Unverified record
V	Verified record
E	Neither U nor V. End-of-record (EOR) is in error.

NOTE: The error description record is in EBCDIC format. For example, a 2 is represented as X'F2'; a C is represented as X'C3'.

A level status of other than zero is presented with error records resulting from the following:

- The start-of-record (SOR) location has a character defined as an error.
- The record contains two or more data check bytes in succession.
- The record is longer than the user-specified maximum length record.
- The length of the record is not equal to the length of the first valid record of the same program level encountered on the MDI cartridge from which data is being obtained.
- The record has a data duplication dependency on a previous record with one of the foregoing.

The level status is set to zero when the editing routine encounters a record without one of the previous errors, a canceled record, or the first record of a cartridge.

Type Status (Byte 1)

The type status indicator identifies records in error because of SOR, EOR, length, field, and/or data check error conditions.

The type status is presented with each error record and has a value of:

0	For any record that has no *identifiable* error(s), but contains questionable data because of a level status of other than zero. (See Level Status.)
1	For any record that has an SOR character of other than P1 through P8 or a GS code; or that has an EOR character of other than a VOK code when the user has specified VERCHK=VOKCHK; or that has an EOR character of other than a VOK code or RM code when the user has specified VERCHK=NOCHK.

2 For any record that has an incorrect length because it is:

- Longer than the specified maximum, or
- Shorter than the specified minimum (MINLN), or
- Not equal to the length of the first valid record of the same program level encountered on the MDI cartridge from which data is being obtained.

4 For any record that has one or more field error(s). A field error is a field where duplication and/or left-zero justification functions did not occur due to an error condition.

8 For any record that has a data check error.

The type status indicator can also have hexadecimal values of 3, 5, 6, 7, 9, A, B, C, D, E and F. These values indicate various combinations of SOR, EOR, length, field, and data check errors. For example, a value of A indicates a record with a data check error (8) as well as an incorrect length (2).

NOTE: A data check error is indicated by the presence of 2772 replacement characters (EBCDIC SUB character, X'3F'), in the input.

Program Level (Byte 2)

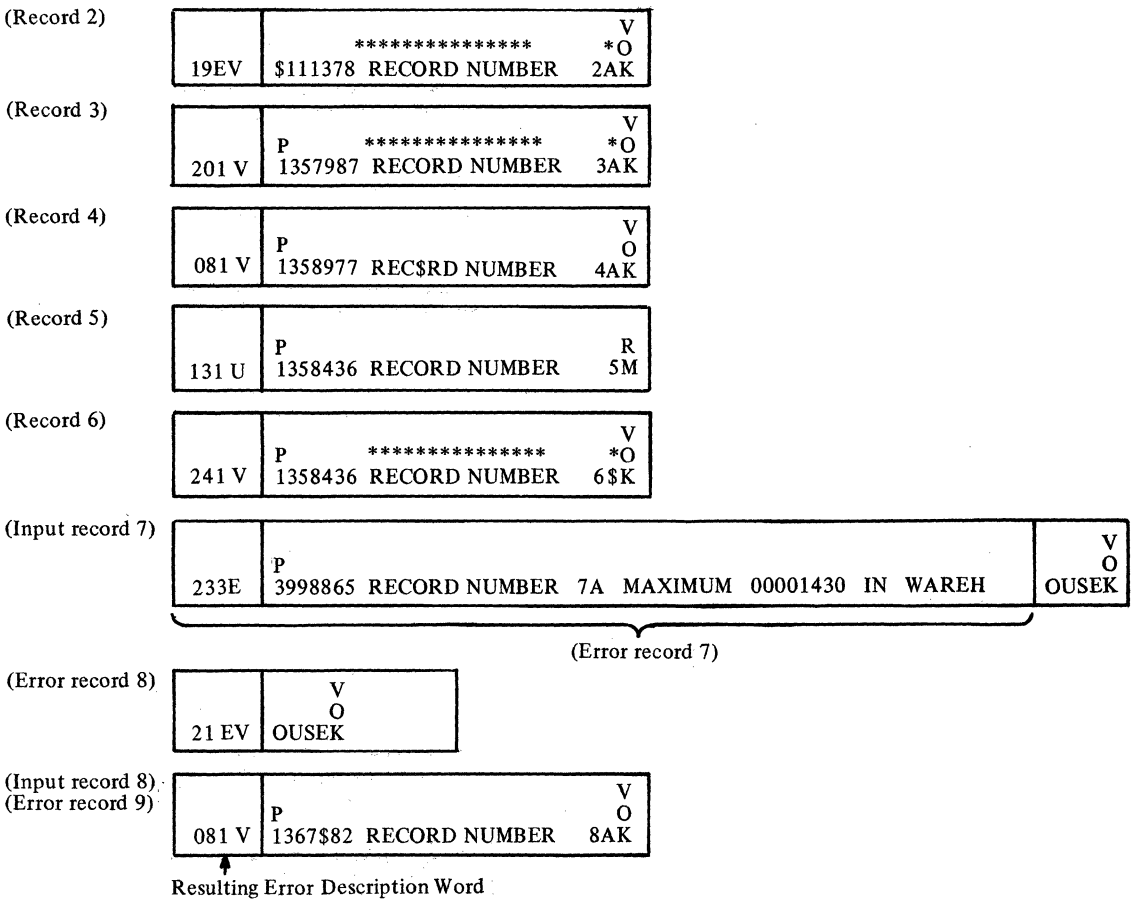
This byte contains an indication of the start-of-record (SOR) character associated with this record.

Record Status (Byte 3)

This byte contains an indication of the end-of-record (EOR) character associated with this record.

Sample Records Containing Errors

These records show some of the errors that may occur during processing and their effect on the error description word (EDW). For these records, the maximum record length is specified as 50; EDITR and VOKCH are specified, and the hexadecimal REPLACE character is '5B' ($). An asterisk in the records indicates the presence of a DUP code in the location before editing.



Record 1 was a valid record. It contained a program level 1 code and thus established the valid length for all program level 1 records received from the cartridge.

Record 2 has a data check in the SOR location. Level status is set to 1 because the SOR location might have contained a cancel code that would cause any data duplicated into the following record to be questionable.

Record 3 has no identifiable error but may contain questionable data because it contained DUP codes and follows a record with a level status of 1.

Record 4 has a data check error. Because it contained no DUP codes, the level status is set to 0.

Record 5 is shorter than first program level 1 records received from the cartridge (length error). This record also contains an RM code rather than a VOK code in the EOR location VOKCHK was specified). Because the editing routine cannot determine why the record is short, all data duplicated from this record is questionable; therefore, the level status is set to 1.

Record 6 contains a DUP code that is beyond the last position of the preceding record.

Record 7 is longer than the maximum specified record length. Note that it is passed as two records. The first record indicates an EOR error and a length error; the second indicates an SOR error.

Record 9 has a data check error. Because it contained no DUP codes, the Level Status is set to zero.

Programming Considerations

- All canceled records are bypassed and are not passed as erroneous records.
- All input records less than three bytes in length (SOR location, one data byte, EOR location) are treated as canceled records. An input record of this size may be the remaining portion of a record that was longer than the maximum user-specified record size.
- Data duplication occurs with the DUP code replaced by the character from the corresponding location of the previous record in the work area when control was last returned to the editing routine.
- Data duplication does *not* occur and the DUP code is replaced with the user-specified error replacement character, and a field error is indicated, for any of the following conditions:

  The DUP code is encountered in the first record of a cartridge.

  The DUP code is encountered in a record and previous record was a canceled record.

  The DUP code is encountered in a record and its position would duplicate the previous record's end-of-record delimiter location or a position beyond the length of the previous record.

- Left-zero justification does *not* occur, the left-zero fill code (LZ) is replaced with the user-specified error replace character, and a field error is indicated, for either of the following conditions:

  The left-zero fill code (LZ), is encountered without its corresponding left-zero start code (LZS).

  The user-specified maximum record size is exceeded before encountering the valid end of a left-zero field.
- For BTAM users using dynamic buffering, the BSC control characters ETB and ETX should not be entered as data on IBM 50 MDI cartridges.

End-of-Cartridge Code

A unique code, written by the IBM 50 MDI, is used to signal to the 2772 control unit that all meaningful data on a cartridge has been read. For the MDI cartridge, the end-of-cartridge code is the ED character (X'19').

After initiation of a Read operation the MDI continues to read data from the tape until it senses the ED character. When the MDI sends this character to the 2772, the 2772 signals the tape to rewind and transmits the data in its buffer to the computer.

Appendix H: Conserving Main Storage

Several operands of the INTRO macro instruction can be used to reduce the amount of main storage required by a TCAM MCP.

If disk queuing is not needed, specifying DISK=NO results in a saving of 140 bytes.

When disk queuing is required, if the application will permit specification of only one channel program block, CPB=1 saves 730 bytes in addition to the actual amount of storage required for the CPBs.

If the system interval is not required, specifying INTVAL=0 will save 660 bytes.

If ENVIRON=TCAM and the system configuration permits, specifying the LINETYP= operand as something other than LINETYP=BOTH results in a considerable saving. The number of bytes saved is:

LINETYP=BISC	2110
LINETYP=STSP	4220
LINETYP=MINI	6720

If ENVIRON= does not specify TCAM, specifying LINETYP=STSP will result in a saving of 4220 bytes.

If the system timer feature is not required, specifying FEATURE=(,,NOTIMER) saves 820 bytes.

If ENVIRON=TCAM, the other suboperands of the FEATURE= operand will also reduce main storage requirements. FEATURE=(DIAL,NO2741) saves 70 bytes. FEATURE=(NODIAL,NO2741) saves 420 bytes. However, FEATURE=(NODIAL,2741) does not result in a saving.

The use of the USEREG=, DTRACE=, OLTEST= and COMWRTE= operands increases the amount of main storage required by the MCP. DTRACE= requires an additional amount of main storage equal to the value specified multiplied by four. OLTEST= requires 1K of main storage for each integer specified (i.e., if OLTEST=10 is coded, an additional 10K of main storage is implied). For a discussion of the effects on storage requirements of COMWRTE=, see the section *Debugging Aids* in *Using TCAM Service Facilities.* For a discussion of the use of the USEREG= operand and its affect upon storage requirements, see *User Code in a Message Handler* in the chapter *Designing the Message Handler.*

The WTTONE= operand will require an additional n+2 bytes of main storage, where *n* is the integer specified for WTTONE=.

In addition to the INTRO operands, operands of the DCB, TERMINAL and TPROCESS macros, in conjunction with INTRO, may have an effect upon the amount of main storage required for the MCP.

If dynamic allocation of buffers is not required, and all DCB macros specify PCI=(N,N), 930 bytes are saved, The queuing type specified on the TERMINAL and TPROCESS macros, in conjunction with the MSUNITS= and DISK= operands of the INTRO macro, changes the amount of main storage required. If reusable disk queuing or multiple queue types are used, an additional 3510 bytes are needed. Disk queuing only saves 3300 bytes over a combination of disk and main-storage queuing. Main-storage-only queuing saves 4110 bytes over a combination of disk and main-storage queuing.

The amount of main storage required may also be reduced by proper specification of buffers and buffer units. Specification is provided in the INTRO operands LNUNITS=, MSUNITS= and KEYLEN=, and the DCB operand BUFSZ=. A checklist for determination of proper size for the application is found in *Design Considerations* in *Defining Buffers.*

The use of these operands depends on the requirements of the application. It may not be possible to utilize all of the savings possible in a particular application.

The operands and number of bytes saved are summarized in the following chart.

MACRO	*OPERAND(S)*	*BYTES*
INTRO	DISK=NO	140
	CPB=1	730
	INTVAL=0	660
	LINETYP=BISC	2110
	LINETYP=STSP	4220
	LINETYP=MINI	6720
	FEATURE=(,,NOTIMER)	820
	FEATURE=(DIAL,NO2741)	70
	FEATURE=(NODIAL,NO2741)	420
	MSUNITS=n,DISK=NO	4110
	MSUNITS=0,DISK=YES	3300
DCB	PCI=(N,N)	930

accepting: the process in which a destination station acquires a message transmitted to it from the central computer. Entering and accepting are functions of a station.

accepting station: a destination station that acquires a message.

access line: a switched line continuously connecting a remote station and the transmission control unit to a switching center (exchange). A telephone number is associated with the access line.

access method: a combination of an access technique (either queued or basic) and a given data set organization (for instance, sequential, partitioned, indexed sequential, or direct) that allows the programmer to transfer data between main storage and I/O devices.

active line: a communication line that is currently available for transmission of data. Contrast with *inactive line*.

active station: a station that is currently eligible for entering and/or accepting messages on the line. A station may be active for entering or active for accepting, or both; the status of a station is determined by the status of the line it is on, by the type of character (+ or -) coded in the invitation list entry for the station, by the presence or absence of a HOLD macro in the outgoing group of the Message Handler handling outgoing messages for this station, and by the five operator commands (ACTVBOTH, ENTERING, NOENTRNG, NOTRAFIC, SUSPXMIT, that directly affect the station's status.

addressing characters: identifying characters, sent by the computer, that cause a particular station (or component) to be selected to accept a message sent by the computer.

answering: a procedure by which a called party completes a connection (for switched lines).

application program: a user-provided program that processes the text portions of messages. Application programs run asynchronously with the Message Control Program, and are usually located in another partition or region of main storage. TCAM application programs are optional; there may be many or none, depending on the needs of the user.

ARU: see *audio response unit*.

audio line: a communication line attached to an audio response unit such as the IBM 7770 Model 3 Audio Response Unit. An audio communication line is always switched.

audio response unit (ARU): a control unit that provides much the same functions for audio stations that a transmission control unit provides for non-audio stations; in addition, it causes an audible response to be made to an audio inquiry.

audio station: a unit of equipment associated with an audio response unit, at which keyed or dialed data is entered for transmission to the computer; an audio response is produced by the ARU as output.

Auto Answer: a machine feature that allows either a transmission control unit or a station to respond automatically to a call that it receives over a switched line.

Auto Call: a machine feature that allows either a transmission control unit or a station to automatically initiate a call over a switched line. A dialing operation that originates at the central computer must use the Auto Call machine feature.

Auto Poll: A machine feature of a transmission control unit that permits it to handle negative responses to polling without interrupting the central processing unit. At the end of the invitation list, polling is resumed automatically at the beginning of the list.

available-unit queue: a queue in main storage to which all buffer units are assigned initially (that is, before allocation to TCAM lines and application programs requiring buffers). *Empty* buffer units (that is, buffer units whose contents have been processed by the incoming or outgoing group of an MH, and that are not assigned to the main-storage message queues data set) are returned to the available-unit queue, from which they are reallocated.

bid: in the contention form of invitation or selection, an attempt by the computer or a station to gain control of the line so that it can transmit data.

binary synchronous communications (BSC): data transmission in which character synchronization is controlled by timing signals generated by the device that originates a message (and the device that obtains the message recognizes the *sync pattern* at the beginning of the transmission--the devices are locked in step with one another); contrast with *start-stop transmission*.

block: that portion of a message terminated by an EOB or ETB line-control character or, if this is the last block in the message, by an ETX or EOT line-control character. When end-of-block checking is specified in the STARTMH macro, messages are checked for certain types of transmission and user-specified logical errors on a block-by-block basis.

BSC: see *binary synchronous communications*.

buffer: an area in main storage into which a message segment is read, or from which a message segment is written. Buffers are temporary data-holding areas that are used to compensate for the difference between the rate at which data can be entered from or accepted by a station and the rate at which it can be processed by the central processing unit; buffers also may be used as work areas in TCAM. The size of TCAM buffers is designated by the user. (See also *hardware buffer*.)

buffer allocation: the assignment of buffers by TCAM to lines or application programs in preparation for reception of message segments from stations on the lines or from application programs. (See also *dynamic buffer allocation* and *static buffer allocation*.)

buffer deallocation: for a sending operation, deallocation consists of returning the units that compose the buffer to the available-unit queue after the data in these units has been sent to its destination station or application program; for a receiving operation, deallocation consists of transferring full buffers from the line or application program to which they were assigned to the incoming group of the MH that is to process the message segments they contain.

buffer prefix: a control area contained within each TCAM buffer. The prefix for the buffer containing the first segment of a message is 30 bytes long, while the prefix for each buffer containing a subsequent segment of the message is 23 bytes long. The user must allow room for the buffer prefix when he

specifies his buffer size. TCAM fills the prefix area with buffer control information.

buffer unit: the basic building block from which TCAM buffers are constructed. All units in a particular TCAM system are the same size; this size is specified by the KEYLEN= operand of the INTRO macro.

buffer unit pool: all the buffer units in a particular TCAM system together constitute the buffer unit pool for that system. The number of units in the pool is equal to the sum of the integers specified by the LNUNITS= and MSUNITS= operands of the INTRO macro.

buffered terminal: a terminal having a hardware buffer. As used in this book, a buffered terminal is an IBM 2740 Model 2 Station or IBM 2770 station whose TERMINAL macro specifies BFDELAY=integer. When the BFDELAY= operand of TERMINAL is coded, messages are sent to the station segment-by-segment; after a segment is sent, the Message Control Program pauses before sending the next segment to allow the station's buffer to empty. During this pause, the MCP may send segments to other stations on the line.

calling: a procedure that establishes a connection over a switched line; a series of electrical signals, corresponding to the telephone number of the station or computer with which contact is to be made, are sent down the line; these pulses or notes cause automatic switching equipment belonging to the common carrier to establish the connection, if the party being called is free to accept the call.

cascade entry: an entry in the terminal table associated with a cascade list.

cascade list: a list of pointers to single, group, or process entries. A message is queued for the valid entry in the list with the fewest messages queued for it.

central processing unit (CPU): a unit of a computer that controls interpretation and execution of instructions.

channel program block (CPB): a TCAM control block used in the transfer of the data between buffer units and message queues maintained on disk. The CPB= operand of the INTRO macro specifies the number of CPBs to be provided in a TCAM system.

checkpoint data set: an optional TCAM data set that contains the checkpoint records used to reconstruct the MCP environment after closedown or system failure, when the TCAM checkpoint/restart facility is utilized.

checkpoint records: records, located in the checkpoint data set, that are used to reconstruct the MCP environment upon restart following closedown or system failure. There are four types of checkpoint records: environment records, incident records, checkpoint request records, and a control record.

checkpoint request record: a checkpoint record taken as a result of execution of a CKREQ macro issued in an application program; the record contains the status of a single destination queue for the application program. The latest checkpoint request record for a message queue is used during restart to cause sending from that queue to the application program to begin with the message that follows the last message sent to the program from that queue at the time the checkpoint request record was taken, rather than with the message following the last message marked serviced.

checkpoint/restart: a TCAM facility that records the status of the teleprocessing network at designated intervals or following certain events. Following system failure or closedown, the checkpoint/restart facility uses the records it has taken to restore the Message Control Program environment as nearly as possible to its status before the failure or closedown.

closedown: an orderly deactivation of the MCP by either an MCPCLOSE macro instruction issued in an application program or an operator command. See *quick closedown* and *flush closedown*.

cold restart: start-up of a TCAM Message Control Program 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 start-up), and is the only type of restart possible when no checkpoint/restart facility is used.

common carrier: in communications, a government-regulated private company that furnishes the general public with telecommunications services; a telephone or telegraph company.

component: an I/O device associated with a station.

computer: in this publication, the central processing unit in which the TCAM Message Control Program is located.

contention: classically, a line-control scheme in which stations on a line compete for the use of that line; the station that is successful in gaining control of the line is able to transmit. In a TCAM system, the term is applied to any point-to-point line configuration when the station on the line does not use polling and addressing characters.

continuation restart: a restart of the TCAM Message Control Program following termination of the Message Control Program because of system failure; the TCAM checkpoint/restart facility is used to restore the MCP environment as nearly as possible to its condition before failure.

control characters: characters transmitted over a line that are not message data, but which cause certain control operations to be performed when encountered by the computer, transmission control unit, or station; among such operations are polling and addressing, message delimiting and blocking, transmission-error checking, and carriage return.

control record: a record, included in a checkpoint data set, that keeps track of the correct environment, incident, and checkpoint request records to use for reconstructing the Message Control Program environment during restart.

CPB: see *channel program block*.

CPU: see *central processing unit*.

DASD: direct-access storage device, also referred to as *disk*. The DASD devices supported by TCAM are the IBM 2311 Disk Storage Drive and the IBM 2314 Direct Access Storage Drive.

data collection: a telecommunications application in which data from several locations is accumulated at one location (in a queue or on a file) before batch processing.

data control block (DCB): an area of main storage that serves as a logical connector between the problem program and a data set. The data control block also can be used to provide control information for any transfer of data. A data control block must be created for each TCAM data set except a message queues data set residing in main storage; a DCB macro instruction is used to create a data control block.

Data set:

1. a named, organized collection of logically related records (program data set). The information is not restricted to a specific type, purpose, or storage medium. Among the data sets specifically related to TCAM are the line group data sets, the message queues data sets, the checkpoint data set, the message log data set, and the input and output data sets for a TCAM-compatible application program.
2. a device containing the electrical circuity necessary to connect data processing equipment to a communication channel; also called a subset, Data-Phone*, modulator/demodulator, or modem.

dead-letter queue: the destination queue for the station or application program named by the DLQ= operand of the INTRO macro instruction. If an invalid destination is detected in a message header by a FORWARD macro instruction, and if no user-exit is specified in the FORWARD macro, that message is sent to the dead-letter queue.

delimiter macro instruction: a TCAM macro instruction that classifies and identifies sequences of functional macro instructions and directs control to the appropriate sequence of functional macro instructions.

descriptor code: under Multiple Console Support, indicates the means of message presentation and message deletion on display devices.

destination: the place to which a message being handled by a TCAM Message Handler is to be sent. A destination may be either a station defined by a TERMINAL macro, a group of stations defined by a TLIST macro, or an application program defined by a TPROCESS macro. One or more destinations may be specified in fields of the message header that are checked by a FORWARD macro, or a single destination may be specified for all messages handled by a particular inheader subgroup by means of the DEST= operand of a FORWARD macro issued in that subgroup.

destination field: a field in a message header containing the name of a station or application program to which a message is directed.

destination queue: a queue on which messages bound for a particular destination are placed after being processed by the incoming group of a Message Handler. A separate destination queue is created for each station defined by a TERMINAL macro specifying queuing by terminal, one for each line whose stations are defined by TERMINAL macros specifying queuing by line, and one for each application-program process entry (defined by a TPROCESS macro) to which the application program may direct GET or READ macros. Destination queues are maintained in message queues data sets which may be located on disk or in main storage. Queuing messages by destination permits overlap of line usage in I/O operations. See also *process queue*.

destination station: a station that accepts a message sent to it by the outgoing group of the Message Handler that is specified for the line to which the accepting station is assigned.

dial: see *calling*.

dial line: see *switched line*.

direct-access storage device: see *DASD*.

disabling the line: a process whereby TCAM causes the computer to condition either the transmission control unit or the audio response unit to ignore incoming calls on a switched line. Once this is accomplished, the line is available for TCAM to send queued messages to a station on that line. See *enabling the line*.

disk: see *DASD*.

distribution entry: an entry in the terminal table associated with a distribution list. A distribution entry is created by a TLIST macro.

distribution list: a list of single, group, cascade, or process entries; when a message is directed to the distribution entry associated with this list, TCAM sends the message to each destination named in the list.

dynamic buffer allocation: the assignment of buffers to a line on an as-needed basis, after a message has started coming in over the line. Dynamic allocation occurs following program-controlled interruptions, and is specified by the PCI= operand of the line group DCB macro. See also *static buffer allocation*.

EBCDIC: Extended Binary Coded Decimal Interchange Code.

enabling the line: a process whereby TCAM causes the computer to condition either the transmission control unit or the audio response unit to respond to incoming calls on a switched line. See *disabling the line*.

end-of-address (EOA) character:

1. a control character or characters transmitted on a line to indicate the end of non-text characters (for example, addressing characters).
2. a TCAM character that must be placed in a message if the system is to accommodate routing of that message to several destinations; the character must immediately follow the last destination code in the message header; and must also be specified by the EOA= operand of the FORWARD macro for the message.

entering: the process in which a station places on the line a message to be transmitted to the central computer (a station *enters* and *accepts* messages, while a computer *sends* and *receives* messages).

environment record: a record of the total teleprocessing environment at a single point in time. The environment record resides in the checkpoint data set; at restart time, an environment record is updated by the contents of incident records that were taken after the environment record was taken, and the updated environment record is then used to reconstruct the Message Control Program environment as it existed before MCP closedown or system failure.

EOA: see *end-of-address character*.

error record: five bytes assigned to each message being processed by a Message Handler; these bytes indicate physical or logical errors that have occurred during transmission on the line or during subsequent processing or queuing of the message, and are checked by error-handling macros in the inmessage and outmessage subgroups of a Message Handler.

---

*Trademark of the American Telephone & Telegraph Co.

error recovery procedures (ERP): a set of internal TCAM routines that attempt to recover from transmission errors.

exchange: a communications switching center.

FEFO (first-ended first-out): a queuing scheme whereby messages on a destination queue are sent to the destination on a *first-ended first-out* basis within priority groups. That is, higher-priority messages are sent before lower-priority messages; when two messages on a queue have equal priority, the one whose final segment arrived at the queue earliest is sent first.

FIFO (first-in first-out): a queuing scheme whereby equal-priority messages on the same destination queue are sent in the order that their first segments arrived at the queue.

flush closedown: a closedown of the TCAM Message Control Program during which incoming message traffic is suspended and queued outgoing messages are sent to their destinations before closedown is completed; this form of termination is known as a *flush* closedown because unsent messages are flushed from the message queues. See also *quick closedown*.

functional macro instructions: TCAM macros that perform the specific operations required for messages directed to the Message Handler. See also *delimiter macro instructions*.

group entry: an entry in the terminal table associated with a group of terminals having the group-code machine feature.

half-duplex: a communication line over which transmissions can occur in either direction, but only in one direction at a time.

hardware buffer: a buffer that is located in a station, as opposed to the buffers for the TCAM MCP, which are located in the computer. The IBM 2740 Communication Terminal Model 2, for example, contains a hardware buffer that accomodates up to 120 characters. See also *buffered terminal*.

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, a priority level for the message, etc. The message header is operated on by macros in the inheader and outheader subgroups of the Message Handler.

header buffer: a buffer containing a header segment.

header segment: a message segment containing all or part of the message header.

identification characters (ID characters): characters sent by a BSC station on a switched line to identify the station. ID characters can also be assigned to the computer; (by the CPUID = operand of the INVLIST macro) in this case, the computer and the station can exchange ID sequences. TWX stations also use ID characters.

idle: describes a line that is not currently available for transmission of data because IDLE was coded in the OPEN macro for the line group data set containing the line. Such a line may be activated by a STARTLINE operator command.

inactive line: a communication line that is not currently available for transmission of data. Contrast with *active line*.

inactive station: a station that is currently ineligible for entering and/or accepting messages. A station may be inactive for entering or inactive for accepting, or both; the status of a station is determined by the status of the line it is on, by a special character (+ or -) coded in the invitation list entry for the station, by the presence or absence of a HOLD macro in the outgoing group of the Message Handler handling outgoing messages for this station, and by the five operator commands (ACTVBOTH, ENTERING, NOENTRNG, NOTRAFIC, SUSPXMIT) that directly affect the station's status.

incident record: a checkpoint record residing in the checkpoint data set on a DASD; an incident record logs a change in station status or in the contents of an option field that occurred since the last environment record was taken. Incident records are used to update the information contained in environment records at restart time after a closedown or system failure.

incoming group: that portion of a Message Handler designed to handle messages arriving for handling by the Message Control Program. See also *outgoing group*.

incoming message: a message being transmitted from a station to the computer.

input: of or related to a message transmission that involves entering data at a station or receiving data at the computer.

input data set: a logical data set for a TCAM-compatible application program. The input data set contains all messages or records being sent to the application program from a single process queue. Though it is not located in a physical medium, the input data set requires a DD statement and a DCB macro for its definition and must be activated and deactivated by OPEN and CLOSE macros. See also *output data set*.

input sequence number: a means of ensuring that messages are received from a source in the correct order; the user may place a sequence number in the header of each message entered by a station or application program, and code a SEQUENCE macro in the incoming group of his Message Handler. The SEQUENCE macro checks the sequence number for each message; if the number is not one more than that assigned to the previous message received from that origin, a bit is truned on in the message error record.

inquiry processing: a TCAM application in which the Message Control Program receives a message from a station, then routes it to an application program that processes the data in the message and generates a reply; the reply is routed by the Message Control Program to the inquiring station. Response time often may be shortened by specifying the lock mode (by a LOCK macro in the Message Handler) and by locating the message queues data set containing the queues for the application program in main storage.

intercepted station: a station to which no messages may be sent. A station is intercepted by issuing a HOLD macro instruction in the outmessage subgroup of a Message Handler; the suspension is either for a specified time interval or until either an operator command or an application program macro instruction is issued to release messages held for the intercepted station.

invalid destination: a destination specified for a message that does not correspond to a valid terminal table entry.

invitation: the process in which the computer contacts a station in order to allow the station to transmit a message if it has one ready.

invitation delay: a period of time (specified by the INTVL = operand of the line group DCB macro), during which outgoing messages are sent to nonswitched polled stations for which receiving has priority over sending (because CPRI = R is coded in the line group DCB macro). This delay is observed for all

such stations on a line when the end of the invitation list for that line is reached. The delay in polling is observed for such stations whether or not the computer has any messages to send them. If no invitation delay is specified for such stations, no messages can be sent to them.

invitation list: a series of sets of polling characters or identification sequences associated with the stations on a line; the order in which sets of polling characters are specified (in the INVLIST macro for the line) determines the order in which polled stations are invited to enter messages on the line.

inward WATS: a service provided by the telephone company, whereby all incoming calls from within a prescribed area are charged to the receiving subscriber at a flat rate. See also *WATS*.

line: the communications medium linking the computer to one or more remote stations; message transmission occurs over this medium. See also *nonswitched line*, *switched line*, *point-to-point line*, and *multipoint line*.

line control: the scheme of operating procedures and control signals by which a telecommunications system is controlled.

line control block (LCB): an area of main storage containing control information for operations on a line; one LCB is maintained by TCAM for each line in the system.

line-control characters: characters that control transmission of data over a line; for example, line control characters delimit messages, cause transmission-error checking to be performed, indicate whether a station has data to send or is ready to receive data.

line group: a set of one or more communication lines of the same type, over which stations with similar characteristics can communicate with the computer.

line group data set: a Message Control Program data set consisting of all the lines in a line group; the messages that are transmitted on these lines constitute the data in this data set. A line group data set is defined by a line group DCB macro instruction, and by a DD statement for each line in the line group.

line group DCB: a data control block created by a line group DCB macro instruction; information in the data control block defines the line group to TCAM.

local station: a station whose control unit is connected directly to a computer data channel by a local cable. See *remote station*.

lock mode: a TCAM facility, invoked in a Message Handler by the LOCK macro, whereby a station entering an inquiry message for an application program is held on the line by the Message Control Program until a response has been returned to it by the application program. Use of the lock mode decreases response time because there are no interruptions on the line before a response is returned. If LOCK is executed and CONV = YES is coded in the STARTMH macro, tete-a-tete interaction (defined in this Glossary) is in effect for the station. A station may be placed in lock mode either for the duration of a single inquiry and response (*message* lock mode) or for the duration of several inquiry-response cycles (*extended* lock mode). The type of lock mode is specified in the LOCK macro.

log: a collection of messages or message segments placed on a secondary storage device for accounting or data collection purposes. The TCAM logging facility is invoked by a functional macro instruction issued in a Message Handler.

log data set: a data set consisting of the messages or message segments recorded on a secondary storage medium by the TCAM logging facility. A log data set is defined by means of a BSAM DCB macro instruction that is issued with the DCB macro instructions defining the line group data sets, the message queues data sets, and the checkpoint data set.

logtype entry: an entry in the terminal table associated with a queue on which complete messages reside while awaiting transfer to the logging medium (a logtype entry is not needed if message segments only are to be logged). A logtype entry is created by a LOGTYPE macro.

MCP: see *Message Control Program*.

message: a unit of data received from or sent to a station that is terminated by an EOT or ETX control character or, if the CONV = operand of the STARTMH macro is coded CONV = YES, by an EOB or ETX control character. A TCAM message is often divided into a header portion, which contains control information, and a text portion, which contains the part of the message of concern to the party ultimately receiving it.

Message Control Program (MCP): a set of user-defined TCAM routines that identify the teleprocessing network to the System/360 Operating System, establish the line control required for the various kinds of stations and modes of connection, and control the handling and routing of messages to fit the user's requirements.

Message Handler (MH): a sequence of user-specified TCAM macro instructions in the Message Control Program that examine and process control information in message headers, and perform functions necessary to prepare message segments for forwarding to their destinations. One Message Handler must be assigned to each line group by the MH = operand of the line group DCB macro, and one must be assigned to each TCAM-compatible application program by the MH = operand of the PCB macro. The incoming group of an MH handles messages received from either an originating station or an application program; the outgoing group of an MH handles messages prior to their being sent to a destination station or application program.

message priority: refers to the order in which messages in a destination queue are transmitted to the destination, relative to each other. Higher-priority messages are forwarded before lower-priority messages. Up to 255 different priority levels may be assigned to a single destination (by the LEVEL = operand of the TERMINAL or TPROCESS macro). The priority for each message sent to the destination may be specified in the message header or assigned by a PRIORITY macro; in either case, a PRIORITY macro should be coded in the inheader subgroup handling the message.

message queue: see *destination queue*.

message queues data set: a TCAM data set that contains one or more destination queues. A message queues 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 queues data sets (one in main storage, one on reusable disk, one on non-reusable disk) may be specified for a TCAM Message Control Program.

message segment: the portion of a message contained in a single buffer.

message switching: a telecommunications application in which a message is received from a remote station, stored until a

suitable outgoing line is available, and then transmittted to its destination station. TCAM message switching can be handled entirely by the Message Control Program.

MH: See *Message Handler*.

multipoint line: a nonswitched line that connects several remote stations to the computer.

nonswitched line: a communication line that links stations for a continuous period, or for regularly recurring periods; also known as a private, leased, or dedicated line.

nontransparent mode: a mode of binary synchronous transmission in which all control characters are treated as control characters (that is, not treated as text). See *transparent mode*.

on-line test (OLT): an optional TCAM facility that permits either a system console operator or a remote-station operator to test transmission control units and remote stations to find out if they work properly.

operator command: a command entered either at an operator control station or at the system console to examine or alter the status of the telecommunications network during execution.

operator control station: a station eligible to enter operator commands. An application program and the system console may also serve as operator control stations. Operator control stations are designated as such by the PRIMARY = operand of the INTRO macro and by the SECTERM = operand of the TERMINAL and TPROCESS macros. See also *primary operator control station* and *secondary operator control station*.

option field: a storage area containing data relating to a particular station, component, line, or application program; certain Message Handler routines that need source- or destination-related data to perform their functions have access to data in an option field. User-written routines also have access to data in an option field. Option fields are defined by OPTION macros and initialized for each station, line, component, or application program by the OPDATA = operand of the TERMINAL or TPROCESS macro.

origin: a station or application program from which a message, or other data originates. See also *destination*.

outgoing group: that section of a Message Handler that manipulates outgoing messages after they have been removed from their destination queues. The outgoing group has three types of subgroup—the outheader subgroup, which executes on outgoing header segments; the outbuffer subgroup, which executes on each outgoing segment; and the outmessage subgroup, which does not execute until after the message has been sent to its destination, if possible. See also *incoming group*.

output data set: a logical data set for a TCAM-compatible application program. The output data set contains the messages or records returned from the application program to the Message Control Program by a process entry in the terminal table. An output data set is defined by a DD statement and a DCB macro, and must be activated and deactivated by OPEN and CLOSE macros. See also *input data set*.

output DCB: a data control block created by an output DCB macro. One output DCB is required for each output data set.

output sequence number: a number placed in the header of a message by TCAM that determines the order in which messages were sent to a destination by the computer. When specified in an outheader subgroup, the SEQUENCE macro causes an output sequence number to be placed in the header of each outgoing message; this sequence number is one greater than the sequence number for the last message sent to this destination. See also *input sequence number*.

point-to-point line: a communication line that connects a single remote station to the computer. It may be either switched or nonswitched.

polling: a non-contention line management method whereby the computer invites remote stations on multipoint nonswitched lines and remote terminals on point-to-point lines to enter messages. The computer contacts stations in the order specified by the invitation list; each station contacted is invited to enter messages.

polling characters: a set of identifying characters peculiar to either a station or a component of that station; a response to these characters indicates to the computer whether the station has a message to enter.

prefix: see *buffer prefix*.

primary operator control station: an operator control station that receives, in addition to the responses to commands entered by it, the operator awareness message (IEA000I, described in *TCAM I/O Error-Recording Facility* in the chapter *Using TCAM Service Facilities* is sent whenever an I/O error occurs and TCAM's error-recovery procedures are unsuccesful in correcting it. The primary operator control station is designated by the PRIMARY = operand of the INTRO macro.

priority: see *message priority* and *transmission priority*.

process queue: a destination queue for an application program (see destination queue). A process queue is defined by a TPROCESS macro.

queue: a set of items consisting of:

1. a queue control block (an area in main storage containing control information for the queue), and
2. one or more ordered arrangements of items (the items may be messages, main storage addresses, etc.).

quick closedown: a closedown of the TCAM Message Control Program that entails stopping message traffic on each line as soon as any messages being sent or received at the time the request for closedown is received are transmitted.

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.

receiving: the process in which the central computer obtains a message from a remote station (the message is *entered* by the station). Receiving and sending are functions of the central computer.

record: a logical unit of data, the length of which is defined by the user through the use of operands of the input or output DCB macro and delimiting characters in the message.

relative line number: a number assigned by the user to a communications line of a line group at system generation time or MCP execution time. If a line group is defined at system generation time by a UNITNAME macro, the lines in the group are assigned relative line numbers according to the order in which their hardware addresses are specified in the UNIT = operand of UNITNAME; the line whose address is specified first is relative line number one, that address specified second is

relative line number two, etc. If a line group is defined at MCP execution time by concatenated DD statements, the order in which the DD statements for the lines in the line group are arranged determines the relative line numbers for the lines. The line whose DD statement appears first is relative line number one, the statement that appears second is relative line number two, etc.

remote station: a station that is connected to a computer data channel through either a transmission control unit or an audio response unit. See also *local station*.

retry: an error recovery procedure in which the current block of data (from the last EOB or ETB) is re-sent a prescribed number of times, or until accepted or entered correctly.

routing code: under Multiple Console Support, indicates the consoles to which the messages should be sent.

segment: the portion of a TCAM message contained in a single buffer.

selection: the process whereby the computer contacts a remote station to send it a message.

sending: the process in which the central computer places a message on a line for transmission to a station (the station accepts the message). Sending and receiving are functions of the central computer.

sequence number: see *input sequence number* and *output sequence number*.

single entry: an entry in the terminal table associated with a single station or station component; one such entry must be created (by a TERMINAL macro) for each station in the TCAM system not defined by a group entry.

start-stop transmission: data transmission in which each character being transmitted is preceded by a special control signal indicating the beginning of the sequence of data bits representing the character, and is followed by another control signal indicating the end of the data-bit sequence (character recognition by the device that obtains the data depends on the presence of these control signals for each character); contrast with *binary synchronous communications*.

static buffer allocation: the assignment to a line, before transmission over that line, of all buffers to be used to contain the transmitted data. When PCI = N or PCI = R is coded in the line group DCB macro, the number of buffers specified by the BUFIN = or BUFOUT = operand of the line group DCB macro instruction is assigned to a line before incoming or outgoing transmission begins on that line; once transmission has started, no more buffers are available to handle the data involved in the transmission.

station: either a remote terminal, or a remote computer used as a terminal.

subblock: that portion of a BSC message terminated by an ITB line control character.

switched line: a communication line on which the connection between the computer and a remote station is established by dialing. Also known as a dial line.

symbol: in assembler language, a character or character string used to represent addresses or arbitrary values. A symbol must meet the following requirements:

1. 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.
2. No blanks or special characters are allowed in a symbol.

system interval: a user-specified time interval during which polling and addressing are suspended on multipoint lines to polled stations. The system interval is specified by the INTVAL = operand of the INTRO macro, and may be changed during TCAM initialization, by a SYSINTVL operator command. The INTERVAL operator command tells TCAM to begin the system interval. The system interval is used to minimize unproductive polling, to minimize CPU meter time, and to synchronize polling on the polled lines in the system. See also *invitation delay*.

TCU: see *transmission control unit*.

telecommunications: any transmission or reception of signals, writing, sounds, or intelligence of any nature, by wire, radio, or other electromagnetic media.

teleprocessing: the processing by a computer of data entered at a remote station.

terminal: a point in a system at which data can enter, leave, or enter and leave. A terminal can also be a control unit to which one or more input/output devices can be attached. See *component*.

terminal table: an ordered collection of information consisting of a control field for the table and blocks of information on each line, station, component, or application program from which a message can originate or to which a message can be sent.

tete-a-tete: a mode of message handling in which a station operating in lock mode is polled by the computer. The station responds with a message that ends with a character permitting selection to continue. The computer sends a response message, from an application program, that the station interprets as a positive response.

text: that part of the message of concern to the party ultimately receiving the message (that is, the message exclusive of the header, or control, information).

text segment: a portion of a message that contains no part of the message header.

transmission: the transfer of coded data by an electromagnetic medium between two points in a telecommunications network.

transmission control unit (TCU): a control unit that serves as an interface between communication lines and a computer for logical operations. The transmission control units supported by TCAM are the 2701 Data Adapter Unit Model 1, the 2702 Transmission Control Model 1, and the 2703 Transmission Control Model 1.

transmission priority: refers to the order in which sending and receiving occur, relative to each other, for a particular station. Transmission priority is specified on a line-group basis by the CPRI = operand of the line group DCB macro. The three transmission priorities possible in TCAM are send priority, equal priority, and receive priority. The exact meaning of each priority depends upon the line configuration and type of station. See also *message priority*.

transparent mode: a mode of binary synchronous transmission in which all data, including normally restricted data-link control characters, is transmitted only as specific bit patterns. Control

characters that are intended to be effective are preceded by a DLE character.

TWX: abbreviation of Teletypewriter Exchange Service, a semi-automatic switching service provided by AT & T for interconnecting public teletypewriter subscribers.

unit: see *buffer unit*.

warm restart: a restart of the TCAM Message Control Program following either a quick or a flush closedown; the TCAM checkpoint/restart facility is used to restore the MCP environment as nearly as possible to its condition before failure. See *restart*.

WATS: abbreviation for AT & T's 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 the Message Control Program by GET or READ macros, and from which messages or records are transferred to the MCP by PUT or WRITE macros. The size of the work area must be specified in the BLKSIZE= operand of the input or output DCB macro associated with the data set whose contents are being transferred to or from the work area. A work area may be defined either statically (by a DC or DS assembler instruction) or dynamically (by specifying locate mode in the MACRF= operand of the input DCB macro).

work unit: the amount of data transferred from the Message Control Program to an application program by a single GET or READ macro, or transferred from an application program to the MCP by a single PUT or WRITE macro. The work unit may be a message or a record (or, for QTAM-compatible application programs, a segment).

6-bit Transcode (hexadecimal equivalents) 389
ABEND
  due to not specifying user error-analysis routine 87
  due to overlaying records on message queues data set 78
  due to uncorrectable I/O error 87
  due to wrapping nonreusable disk during flush closedown 79
  exit
    specifying address for problem program 67
  space requirements for attached subtasks 365
  specifying user exit 87-90
  TCAM formatted dump 968
  user routine options 90
accepting 15
accounting on a log data set 87
ACT operand 283
activating and deactivating the MCP 91-107
  sample code 108
active entries
  displaying for invitation list 300
  specifying in invitation list 278
active invitation list identification 278
ACTVATED operator command 300
ACTVBOTH operator command 300
ADDR= operand 32
address operand 286
address vector table 91
  dump 344
addressing characters 24, 36
  for a component 36
  specifying 32
addressing timeout exceeded indicated on message error record 372
ALTDEST= operand
  TERMINAL macro 33
  TPROCESS macro 42
alternate destination
  effect of size of reusable disk data set 78
  queuing requirement 76
  restriction 78
  sending messages to 192
  specifying 33
    for a component 36
    in the terminal table 42
application program 233-293
  abnormal termination 293
  buffers 247-249
    design considerations 247-249
    definition checklist 247
  CKREQ macro 290
    example 291
  CLOSE macro 251
  coordinating restart with MCP 293
  coordinating TCAM and OS checkpoints 289-293
  data set definition 237-246
  data transfer
    BSAM/TCAM completion codes 269
    CHECK macro 269
    coding TCAM macros 263-271
    GET macro 264
    PUT macro 265
    READ macro 265
      multiple 266
    WRITE macro 267
  delimiting record for 43
  entering operator commands from 299
  error exits 271-273
  examining a queue control block from 280-281
  incident checkpoint records of option fields 151
  input data control block 236
  input data set 236
  input DCB macro 238
    format of position field 241
  inquiry/rapid response 286-289
  interface with MCP 235-236
    activating 249-252
    defining components 236-249
    specifying 41
  limiting number of messages sent to 182
  locking to station 165
  MCPCLOSE macro 251-252
  message flow to 235
  message queues for
    recording status of 334
    specifying where maintained 42
  message retrieval 285-286
    POINT macro 285
  moving invitation-list contents to work area 276-279
  moving terminal-table contents to work area 274-276
  network control 273-285
    ICHNG macro 282-284
    ICOPY macro 276-279
    macro summary 273
    MRELEASE macro 284-285
    operator commands (see network control macro descriptions)
    QCOPY macro 280-281
    TCHNG macro 281-282
    TCOPY macro 274-276
  OPEN macro 249-250
  OS checkpoint restriction 290
  output DCB macro 242-245
    format of position field 244
  password for specific macros 99
  PCB macro 245-246
  POINT macro 285
  priority 233
  process entry 236
  replacing contents of a terminal table entry 281-282
  replacing contents of option fields 281-282
  represented in the terminal table 25
  specifying address of MH for 246
  specified as a secondary operator control station 42
  specifying maximum destination queues used simultaneously 78
  SYNADAF macro 273
    format of TCAM/SAM message buffer 273
  SYNAD exit 271
  TCAM/SAM compatibility 289-293
  testing in non-teleprocessing environment
  work area 235, 253
    defining 253,257
    specifying size 239, 243
    static definition of 253
  work unit 235, 243
    specifying 257
areaname operand
  GET macro 264
  ICHNG macro 283
  ICOPY macro 279
  PUT macro 265
  QCOPY macro 281
  READ macro 267
  TCHNG macro 282
  TCOPY macro 276
  WRITE macro 268
assembler language conventions 13
assembling the MCP 206
AT operand 173
AT&T TWX Terminals
  device-dependent considerations 411-413
  line codes for 400
AT&T 83B3 Terminal
  device-dependent considerations 413
  line code for 399
ATTACH macro considerations 365
attached subtasks
  listed 365
  optional
    checkpoint 330-341
    COMWRITE 346-349, 100
    on-line test 356-358
  required

operator control 295-300
audio terminals
specifying invitation message 20
use of LOCK macro 165
use of TERMINAL macro 37
Auto Answer 51
Auto Call 51
Auto Poll 49
determining if line eligible for 278
determining use on a line 278
switching to 302
verifying use on a line 320
AUTOSTOP operator command 301
AUTOSTRT operator command 302
available-unit queue 57
channel program block work area assigned to 74
channel program block work area replacement 74

base registers for an MH 145
BFDELAY = operand 33
bidding 18
binary synchronous transmission 16 (*see also* BSC)
transparent mode 17
treatment of line control characters 17
BLANK = operand
INITIATE macro 164
LOCK macro 166
MSGEDIT macro 175
MSGTYPE macro 184
PATH macro 188
PRIORITY macro 191
SCREEN macro 195
SETEOF macro 198
SETSCAN macro 200
UNLOCK macro 203
BLKSIZE = operand
input DCB macro 239
log DCB macro 87
output DCB macro 243
block 109
block size 34
component 37
overriding for a component 37
overriding with the MSGFORM macro 34
specifying for messages in transparent mode to a component 37
specifying for outgoing messages in transparent mode 34
BLOCK = operand 180
blocked work units
input data set 240
output data set 244
blocking factors
component 37
overriding for outgoing messages 179
specifying 17
specifying for outgoing messages 179
nontransparent mode 34
BREG = operand 145
BSAM DCB macro operands for a log data set 87
BSAM/TCAM completion codes 269
BSC
device-dependent considerations 411-417
general 415-417
sending and receiving 413-415
input format error indicated on message error record 371
buffered station 48
BUFFER = operand 421
specifying delay between blocks 33
buffers
allocation 62-63
application program 247-249
definition checklist 247
design considerations 247-249
control area 55
deallocating empty units on 147
defining
application program 247-249
MCP 55-64
design considerations 61-64
determining number for read-ahead queue 248
general 63
number of units 61
size 61-62
dump 349
activating 305
dynamic deallocation (effect on EOB checking) 143
header 55
identical characters indicated on message error record 370
checking for 155
identifying incoming subgroup to handle 146
initial and maximum number per line 63
MCP definition checklist 59
message error record indication of insufficient number 370
message header
using scan pointer 118-122
message
format of TCAM/SAM SYNADAF 273
outgoing message
identifying subgroup to handle 148
overriding size specified on the line group DCB macro 33
overriding size specified for a component 37
prefix 55
reserving bytes for inserting date/time/sequence number 69
sample format 56
size 61
for line group 55
overriding 55
specifying for handling messages for an application program 246
specifying for line 68
specifying for MCP when messages are for application program 240
specifying for messages destined for logging medium 45
specifying for messages to MCP from application program 244
specifying for outgoing messages 33, 37
specifying bytes for date/time/sequence number 246
specifying initial number for GET/READ operations 246
specifying initial number for PUT/WRITE operations 246
specifying initial number for receiving 67
specifying maximum number for lines 68
specifying printed debugging listings of 352
example 353
specifying program-controlled interruptions for allocation/deallocation of 69
structure 55-56
text 55
translation of data in 151
unit 55
allocation 58
as a work area in a channel program block 74
deallocation 147
specifying maximum used simultaneously for main-storage queuing 96
specifying number for segments 96
specifying size 95, 96
unit pool 55, 57-58
main-storage message queues data set in 79
BUFIN = operand
line group DCB macro 67
PCB macro 246
BUFL = operand
input DCB macro 240
output DCB macro 244
BUFMAX = operand 68
BUFOUT = operand
line group DCB macro 67
PCB macro 246
BUFSIZE = operand
line group DCB macro 68
LOGTYPE macro 45
PCB macro 246
TERMINAL macro 33

burst mode restrictions 361
bus out
specifying intensive-mode error recording for 328
check indicated in message error record 372

calling
between the computer and a switched station 51-53
busy lines 52-53
specifying time for computer-initiated calls 33
canceling messages 150
CANCELMG macro 150
restrictions 150
specifying conditional execution 150
specifying logical connection between mask and message error record 150
cascade list 430
defining in the terminal table 40
entry 25
specifying actual entry in the terminal table 40
CE mode 361
central computer 15
channel control check indicated on message error record 371
channel data check indicated on message error record 371
channel error indicated on message error record 371
channel program blocks 74-75
determining appropriate number 75
formula for determining initial number of 74
free pool 74
specifying number of 94
channel program codes in operator awareness message 329
channel status word indication of I/O error 326
characters
checking incoming buffers for identical 155
data link control 17
inserting and removing for messages 168-179
parity error indicated on message error record 372
character sets 375
CHECK macro 269
specifying name of data event control block 270
checklists
application program buffer definition 247
checkpoint/restart 331-332
MCP buffer definition 59-60
specifying message queues data sets 82
checkpoint
coding requirements for obtaining 338
coordinating TCAM and OS 289-293
example 220-231
using DCB exit for 292
environment records
example using 336-338
specifying maximum time between 97
exit restriction 289
how it works 333-338
incident record
example using 336-338
specifying 151
incident records
operator commands causing 326
making resident 373-374
module names and sizes 374
OS restriction 290
queuing requirements 80
specifying as additional feature 101
types of records 333-335
checkpoint data set 85-86
DD statement 86
example of opening 103
example of updating environment records 336-338
formula for determining size 338-339
specifying 83, 85
as input/output 102
specifying number of checkpoint request records for 98
checkpoint DCB macro 85
checkpoint request records 334
automatic incident record when data set full 333
priority used in dividing space for 339
specifying data set for 86
specifying number for checkpoint data set 98
checkpoint/restart 330-341
checklist 331-332
CHECKPT macro 151
specifying incident checkpoint records 151
CIB= operand 94
CINTVL= operand 33
CKPTSYN= operand 42
CKREQ macro
checkpoint/restart operands 332
initiating checkpoint request records 334
sample use of 291
specifying maximum number of destination queues used simultaneously for application programs 98
using for checkpoint coordination 290-292
CKREQS= operand 98
CLOCK= operand 33
CLOSE macro
application program 251
specifying data control blocks 251
MCP 106
specifying data control blocks 107
closedown 430, 251-252
automatic environment checkpoint record during 333
flush 181
abnormal termination by wrapping nonreusable disk 79
cold restart following 340
specifying percentage of nonreusable disk records to be used prior to 84
warm restart following 340
initiating through operator control 324
quick 101
cold restart following 340
response to inquiring station in lock mode 287
warm restart following 340
restarting from 330
specifying type 252
code
charts 375-402
control characters 377
conventions used in 375
format of 375
general notes 377
nonequivalent characters 376
substitutions 376
EBCDIC 387
hexadecimal equivalents of 6-bit Transcode 389
invalid destination indicated on message error record 370
line
AT&T TWX 400
AT&T 83B3 399
IBM 1030 390
IBM 1050 391
IBM 1053 printer 393-394
IBM 1060 392
IBM 2260 (Remote)/2265 393-394
IBM 2740 395
IBM 2741 (Correspondence) 398
IBM 2741 (EBCD) 397
IBM 2741 (hexadecimal equivalents of BCD) 396
IBM World Trade Telegraph ITA2 401
IBM World Trade Telegraph ZSC3 402
WU 115A 399
list of translation tables provided 138
TCAM macro formats 367
USASCII 388
CODE macro 151-153
effects on PRIORITY macro 190
format 153
specifying type of translation 153
CODE= operand 182
cold restart 340
automatic recording of changes in option fields 334
building 'Good Morning' message 104-105
following abnormal flush closedown 79
good morning routine gaining control following 340
reformatting DASD message queues data sets 75
COMEDIT printing utility 352
examples of invoking 353

PARM = options 352
command input block (specifying) 94
command reject
indicated on message error record 371, 372
specifying intensive-mode error recording for 328
COMP = operand 36
completion codes for BSAM/TCAM 269
component definition 36-37
component entry 25
computer ID sequence 20
COMWRITE data set 346-347
printing contents of 352
requirements and format 347
specifying 100
COMWRTE = operand 100
conchars operand 124
INITIATE macro 164
LOCK macro 166
MSGTYPE macro 184
PATH macro 188
PRIORITY macro 191
SCREEN macro 195
SETEOF macro 198
UNLOCK macro 203
configurator for on-line test 356
CONNECT = operand
CANCELMG macro 150
ERRORMSG macro 157
HOLD macro 163
MSGGEN macro 181
REDIRECT macro 193
connection error indicated on message error record 371
conserving main storage 427
CONT = operand 144
contention 18, 24
contention line 15
continuation restart 340
replaced by cold restart due to faulty checkpoint records 340
control characters 16
in multiple buffer headers 130
listed 377
removing 110
sending 111
using to vary path of message in MH 129
control information
buffers 55
channel program blocks 74
CONTROL = operand 95
control record 333
control unit errors indicated on message error record 371, 372
control units supported 361
CONV = operand 144
conversational mode 286
conversion of QTAM application programs 403
counter-overflow record 328
access to 329
COUNTER macro 154
specifying location of count field 154
CPB = operand 94
CPINTVL = operand 97
CPRCDS = operand 97
CPRIOPCL operator command 303
CPRI = operand 67
CPUID = operand 20
cross-buffer execution 120
cross-buffer processing
example 121
limitations 121
cross-reference table 345
specifying number of entries in 99
CROSSRF = operand 99
cutoff indicated on message error record 370
CUTOFF macro 155
example 155
restriction 155
specifying maximum characters for messages 155
using the ERRORMSG macro with 155

DASD
checkpoint data set record types 333-335
message queues data set
designing for high message traffic 78
dump 353
preformatting 365
reusable data sets 76-78
sample JCL for obtaining printed output of 353
specifying as input/output 102
specifying PARM = parameters for printed output 354
specifying for message queues data set 94
data check
indicated on message error record 372
specifying intensive-mode error recording for 328
data collection by logging messages 341
data control block 430
checkpoint 85
dump 344
input 238
line group 65
message queues 83
output 242
specifying address
GET macro 264
PUT macro 265
READ macro 267
WRITE macro 268
specifying exit address for problem program 67
specifying for line group 31
specifying name for data set being closed 107
application program 284
specifying name for log data set 45
specifying name for opening application program data set 250
specifying on POINT macro 286
data event control block (specifying name)
CHECK macro 270
READ macro 267
WRITE macro 268
data file (delimiting) 198
data link control 15
data operand 173
data set
application program
defining 237-246
specifying name of data control block for closing 251
specifying name of data control block for opening 250
checkpoint 85-86
DD statement for 86
example of opening 103
example of updating environment
records 336-338
formula for determining size 338-339
specifying 83, 85
specifying as input/output 102
specifying number of checkpoint request records for 98
types of records 333-335
closing 107
COMWRITE
printing contents of 352
specifying for output 100
DASD message queues
preformatting 75
specifying as input/output 102
disk (example of writing debugging information to) 349-352
disk message queues
example of assigning relative record numbers 18
impending failure indicated on message error record 370
impending fullness indicated on message error record 370
input
DCB macro 238-241
specifying 102
specifying format and characteristics of work unit 240
input and output DD statements 245
line group 65-72
creating 65-72

examples of opening 103
guideline for specifying as input/output 103
identifying organization 66
restriction for coding as output 102
specifying activation of line 103
specifying DD statements 70-72
log 87, 344
example of opening 103
list of operands for specifying for BSAM 87
specifying as output 102
specifying name of data control block 45
specifying where messages to be queued 45
logging messages sequentially 341-344
main-storage message queues
destination queue in 79
providing warning when nearly full 96
specifying maximum units used simultaneously 96
main-storage-only queues (lost message indication) 370
MCP
defining 65-90
initialization and activation 101
message queues 72-84
DCB macro 83-84
DD statement 84
destination queues on disk 73
disk efficiency 75
dump 353
examples of opening 103
main storage 79-81
preformatting disk 365
sample JCL for obtaining printed output 353
scanning queues in 335-338
specifying 83, 85
specifying one or more 81-84
specifying PARM = parameters for printed output 354
specifying when user informed that data set no longer crowded 96
specifying whether on a DASD 94
output
COMWRITE requirements and format 347
DCB macro 242-245
specifying 102
specifying format and characteristics of work unit 244
reusable disk queues 76-78
automatic environment checkpoint at zone changeover 333
designing for high message traffic 78
reorganizing 76
reusable or nonreusable disk destination queue (message retrieval) 285
specifying type 102
SYS1.LOGREC error records 329
data
moving between input and output work areas 254
transferring between MCP and application program 252-271
translation of 151
transmission failure indicated on message error record 370
DATA = operand 158
data transfer
BSAM/TCAM completion codes 269
CHECK macro 269
coding TCAM macros in an application program 263-271
GET macro 264
issuing multiple READ macros 266
PUT macro 265
READ macro 265
WRITE macro 267
data-link control characters 17
date
format for inserting in header 155
reserving bytes in buffer for inserting 69, 246
specifying whether to be inserted in header 156
DATE = operand 156
DATETIME macro 155
example 156
reserving bytes for date/time 246
restrictions 156
specifying whether date to be inserted 156
specifying whether time to be inserted 156
DATOPFLD operator command 304
DCB exit
checkpoint restriction 289
using for coordination 292
DCB macros
BSAM operands for specifying a log data set 87
checkpoint 85
list of pertinent checkpoint/restart operands 331
specifying data set 85
specifying ddname 86
specifying problem program exit list 86
specifying use of GET and PUT macros for access 86
input 238-241
designating control of message transfer 239
format of position field 241
identifying data set organization 239
list of pertinent checkpoint/restart operands 332
operands 239-241
operands optionally provided by alternate source 245
specifying ddname 239
specifying EODAD address 241
specifying format and characteristics of work units for input data set 240
specifying optional fields of work unit 240
specifying problem-program exit list 241
specifying size of MCP buffers sent to application program 240
specifying size of record plus optional fields 240
specifying size of work area 239
specifying SYNDAD address 241
specifying type of access to destination queue 239
summary of work-unit formats 258
line group 65-70
format 66
identifying data set organization 66
operands 66-70
specifying buffer size 68
reserving buffer space for date/time/sequence numbers 69
specifying ddname 67
specifying initial buffers for receiving 67
specifying initial buffers for sending 67
specifying invitation delay 67
specifying maximum buffers for lines 68
specifying name of MH 69
specifying name of special characters table 70
specifying names of invitation lists 68
specifying problem-program exit list 67
specifying program-controlled interruption 69
specifying translation table 69
specifying transmission priority for line 67
specifying use of GET and PUT macros for access 66
message queues 83-84
specifying data set 83
specifying ddname 84
specifying either reusable or nonreusable disk 84
specifying percentage of nonreusable disk records to be used before flush closedown 84
specifying problem program exit list 84
specifying use of GET and PUT macros for access 84
output 242-245
designating control of message transfer 242
format of position field 244
identifying data set organization 242
list of pertinent checkpoint/restart operands 332
operands 242-245
operands optionally provided by alternate source 245
specifying ddname 243

specifying format and characteristics of work units for output data set 244
specifying method of transferring messages to destination queue 243
specifying optional fields for work unit 243
specifying problem program exit list 244
specifying size of buffers for messages to MCP 244
specifying size of work area 243
specifying size of work unit plus optional fields 243
specifying SYNAD address 244
summary of work-unit formats 259
dcbname operand
CLOSE macro
application program 251
MCP 106
GET macro 264
LOG macro 168
LOGTYPE macro 45
OPEN macro
application program 250
MCP 102
POINT macro 286
PUT macro 265
READ macro 267
WRITE macro 268
DCB= operand 31
DD statement
checkpoint data set 86
message queues data set 84
specifying for input and output data sets 245
specifying for line group data set 71-72
specifying name of for data control block
input DCB macro 239
line group DCB macro 67
output DCB macro 243
DDNAME= operand
checkpoint DCB macro 86
input DCB macro 239
line group DCB macro 67
log DCB macro 87
message queues DCB macro 84
output DCB macro 243
DEACT operand 283
deactivating the MCP 105-107
CLOSE macro 106
sample activation and deactivation 107
TCAM with application programs 106
TCAM with no application program 105
types of closedown 105
dead-letter queue specification 97
deallocation 147
DEBUG operator command 305
debugging aids 344-355
activating 305
basic coding requirements 355
buffer dump 349
COMEDIT printing utility 352
cross-reference table 345
specifying number of entries in 99
dispatcher subtask trace table 347
dumps 344
error messages 344
example of writing to disk data set 349-352
line I/O interrupt trace table 345
specifying number of entries in table 99
specifying point in routine to gain control when table full 99
message logging 341
non-teleprocessing TCAM applications 233
specifying COMWRITE output data set 100
STCB trace table 347
specifying number of entries in 100
tracing flow of messages 342
writing on data set for later printing 346
decbname operand
CHECK macro 270
READ macro 267
WRITE macro 268
dedicated line 15
delay
changing duration for polling 317
specifying for invitation 67
delimiter
destination field in header for multiple routing 161
inserting for record 178
in application program 43
invitation list 279
MH macros 140-149
functions in MH 118
variable and undefined records 258
destination
expediting transmission of messages to 163
logging 342
maintaining count of outgoing messages for a station 154
message (specifying on FORWARD macros) 160
queue in main-storage message queues data set 79
specifying additional 193
specifying for error message 158
specifying station to receive intercepted messages 285
specifying type of access to queue 239
specifying user-written routine to gain control when invalid 161
destination code error indicated on message error record 370
destination field in optional fields in work area 257
destinations (canceling messages to multiple) 150
DEST= operand
ERRORMSG macro 158
FORWARD macro 160
REDIRECT macro 193
device-characteristics fields in terminal table 275
device-dependent considerations 407-426
BSC devices 413-417
general 415-416
IBM 2770 417
IBM 2780 417
sending and receiving 413-415
start/stop devices 407-413
AT & T 83B3 413
AT & T TWX 411-413
IBM 1030 407
IBM 1050 407
IBM 1060 407
IBM 2260 (Local) 408
IBM 2260 (Remote) 407
IBM 2265 409
IBM 2740 409-410
IBM 2740 basic 409
IBM 2740 basic dial 410
IBM 2740 model 2 410
IBM 2740 on switched line 410
IBM 2740 with station control or station control with checking 409
IBM 2740 with transmit control or transmit control with checking 409
IBM 2741 410
IBM 2760 410
IBM 7770 411
IBM World Trade Telegraph 411
TPEDIT macro for the IBM 50 417-429
device
malfunction indicated on message error record 372
not on control unit indicated on message error record 372
not-ready state indicated on message error record 372
test mode indicated on message errror record 372
device support listed 362-364
dial line as additional feature 101
dial-out option as additional feature 101
DIALNO= operand 32
direct-access storage device specified for message queues data sets 194
disconnection error indicated on message error record 371
disk data set for debugging information 349-352
disk queuing 72-79
advantage of combining checkpoint coordination methods 293
nonreusable 79
specifying as additional feature 101

disk
I/O-error records 327
marking message serviced 335
message retrieval from destination queues 285
nonreusable
specifying for message queues data set 84
specifying percentage of records to be used before flush closedown 84
preformatting message queues data set 365
reusable
automatic environment checkpoint of message queues at zone changeover 333
specifying for message queues data set 84
writing I/O error records to 327
writing permanent error record on 326
DISK = operand 94
dispatcher subtask trace table 347
activating 305
examples of obtaining printed output from 353
specifying number of entries in 100
dispatcher records of subtask activation 347
distribution list
defining in the terminal table 40
entry 25
specifying actual entry in the terminal table 40
DLE 17
DLQ = operand 97
DPRIOPCL operator command 307
DSECOPCL operator command 307
DSORG = operand
checkpoint DCB macro 85
input DCB macro 239
line group DCB macro 66
log DCB macro 87
message queues DCB macro 83
output DCB macro 242
DTRACE = operand 100
dummy invitation list 68
dump of message queues data set 353
sample JCL for printed output 353
specifying PARM = parameters for printed output 354

EBCDIC code 387
translation to line code 151
EDIT = operand 418
end-of-day record 328
access to 329
end-of-file
signaling an application program 262
specifying EODAD address on input DCB macro 241
end-of-message signal for an application program 262
ENTERING operator command 308
entering 15
entries
displaying active (in invitation list) 300
displaying inactive (for a line) 311
specifying total number in invitation list 278
entry
deactivating for a station 313
invitation list example 20
specifying length in invitation list 278
terminal 37
activating nonswitched station for entering 308
terminal-table
count of Start I/O commands 327
count of temporary errors 327
types defined 24
environment checkpoints (specifying maximum time between) 97
environment records 333
example 336-338
how updated 334
specifying number kept in checkpoint data set 97
EOA character 16
example using 110
removing from incoming messages 143
EOA = operand 161
EOB checking
effects of dynamic buffer deallocation on 143
effects of INITIATE macro on 143
when performed 142
EOB completion handling 144
EOB line control character 16, 253, 259
removing from incoming message 143
EODAD address specification on input DCB macro 241
EODAD = operand 241
EOF
signaling an application program 262
specifying EODAD address on input DCB macro 241
EOF message indication 198
EOM 377
signaling an application program 262
EOT line control character 16, 253, 259
EOT = operand 19
equipment check
indicated on message error record 372
specifying intensive-mode error recording for 328
ERRECORD operator command 308
guidelines for using 328
ERROPT = operand 419
error
counter overflow record written on disk 228
end-of-day record written on disk 328
I/O record types 327-328
indicated on message error record 369-372
intensive mode recording 328
displaying current status of 322
irrecoverable 326
terminating connection with station due to 326
logical
requirements for EOB-checking when user-specified 142
testing for 144
permanent I/O record written on disk 327
recoverable
operator awareness message indicating failures 329
retrying the block for 327
suspending transmission to station due to 140
temporary
counter in terminal-table entry for 327
intensive-mode recording for 308
I/O record written on disk for 328
transmission
requirements for EOB checking 142
types for which intensive mode may be specified 328
undefined
indicated on message error record 371
error bits described for message error record 369-372
error codes returned by TCAM Open routines 88-89
error exits for application programs 271-275
error handling 116
error message
generating an unqueued 180
sending when errors occur 157
specifying actual text of 158
user-specified 156
ERRORMSG macro 157-159
format 157
restrictions 157
specifying conditional execution 158
specifying destination for error message 158
specifying error message 158
specifying user-written routine to complete error message processing 159
error record 431
gaining access to 329
I/O 327-329
specifying temporary I/O 308
error-recovery procedures 432
I/O 326
ETB line control character 16, 17, 253, 259
removing from incoming messages 143
ETX line control character 16, 17, 253, 259
removing from incoming messages 143
EXEC statement for passing information to user code 136
execute form on the OPEN macro 103
executing an MCP (sample JCL) 206
exit
DCB

checkpoint restriction 289
using for checkpoint coordination 292
SYNAD 271
user-written routine for invalid destinations 161
exit list for problem program
specifying address on checkpoint DCB macro 86
specifying address on line group DCB macro 67
specifying address on input DCB macro 241
specifying address on message queues DCB macro 84
specifying address on output DCB macro 244
EXIT = operand
ERRORMSG macro 159
FORWARD macro 161
EXLST = operand
checkpoint DCB macro 86
input DCB macro 241
line group DCB macro 67
message queues DCB macro 84
output DCB macro 244
extended lock 287
removing station from 202
EXTEND operand 166

failure of system
cold restart following 340
example of scanning message queues during restart 337
restarting from 330
scanning message queues during restart 335
specifying type of restart for 98
suggestions for establishing checkpoint coordination 291-292
features
specifying additional 101
specifying to conserve main storage 427
FEATURE = operand 101
fieldname operand 181
field addressability requirements 131
file updating sample program 220-231
fixed-format work unit 257
fixed-length work units
input data set 240
output data set 244
specifying size 243
flush closedown 101
ABEND due to wrapping nonreusable disk 79
automatic environment checkpoint record during 333
cold restart following 340
initiating through operator control 324
specifying in application program 252
specifying percentage of nonreusable disk records to be used prior to 84
warm restart following 340
FLUSH operand 252
format error indicated for BSC input on message error record 371
formatting TCAM macros 367
conventions used 367
FORWARD macro
format 160
restrictions 161
specifying destination for messages 160
specifying end of destination fields 161
specifying exit routine to gain control for invalid destinations 161
free pool queue for channel program blocks 74
function modification 118
function operand 172
functional macros 109, 150-204
conditional execution 129

GENERATE macro modified for TCAM 364
GET macro 264
specifying address of data control block 264
specifying address of work area 264
specifying initial buffers to handle data obtained by 246
specifying to gain access to checkpoint data set 86
specifying to gain access to line group data set 66
specifying to gain access to message queues data set 84
GMMSG = operand 104
"Good Morning" message for initial start 104-105
good morning routine gaining control 340
GOTRACE operator command 310
group 111, 113
incoming
required delimiter macro 145, 147
translating to EBCDIC 152
message flow within 123
outgoing required delimiter macro 149
group entry 24
DSECT of 275
group operand 172
grpname operand
ICHNG macro 283
ICOPY macro 279

header 109
controlling path of through an MH 183
format for inserting date and/or time 155
format of field for input sequence number 196
incoming segment records 167
message 110-111
multiple routing delimiter considerations 161
multiple-buffer
considerations for user code in an MH 131
handling 124
processing across buffer 128
origin field validity 185
outgoing segment records 168
scan pointer used for 118-121
header buffer 55
header field
locating 132
header-only message 109
header-processing
functions 111
HOLD macro 162
restrictions 162, 163
specifying conditional execution 162
specifying duration of hold 163
specifying type of hold 163
hold/release 139

I/O device generation 364
I/O error
counter overflow record written on disk 328
end-of-day record written on disk 328
permanent record written on disk 327
recording 327-329
record types 327-328
recovery procedures 326
specifying records of 308
temporary 327
record written on disk 328
I/O interrupt trace facility 345
activating 310
deactivating 314
IBM 50 Magnetic Data Inscriber (MDI) 417
TPEDIT macro for 417-426
IBM 1030 Data Collection System
device-dependent considerations 407
line code for 390
IBM 1050 Data Communication System
device-dependent considerations 407
line code for 391
IBM 1053 Printer line code 392-394
IBM 1060 Data Communication System
device-dependent considerations 407
line code for 392
IBM 2260 (Remote)/2265 Display complex line code 393-394
IBM 2260 Display Station device-dependent considerations
Local 408
Remote 407
IBM 2260 Display Station line-address characters 194
IBM 2265 Display Station device-dependent considerations 409

IBM 2311 Disk Storage Drive 72, 85
IBM 2314 Direct Access Storage Facility 72, 85
IBM 2740 Communication Terminal
device-dependent considerations 409-410
basic 409
basic dial 410
station control or station control with checking 409
switched line 410
transmit control or transmit control with checking 409
line code for 395
IBM 2740 Model 2 Communication Terminal
device-dependent considerations 410
specifying intensive-mode error recording for unusual leading graphic response 328
transmission priority 49
IBM 2741 Communication Terminal
device-dependent considerations 410
line code 396-398
BCD hexadecimal equivalents 396
correspondence 398
EBCD 397
specifying as additional feature 101
IBM 2760 Optical Image Unit device-dependent considerations 410
IBM 2770 Data Communications System
device-dependent considerations 417
transmission priority 49
IBM 2780 Data Transmission Terminal device-dependent considerations 417
IBM 7770 Audio Response Unit device-dependent considerations 411
IBM World Trade Telegraph (WTTA) Terminals
device-dependent considerations 411
line code
ITA2 401
ZSC3 402
ICHNG macro 282-284
restriction 289
specifying line group for modifying invitation list 283
specifying modification or type of modification 283
specifying password 99, 284
specifying relative line number for modifying an invitation list 283
ICOPY 276-279
restriction 289
specifying line group containing invitation list to be displayed 279
specifying relative line number to display invitation list 279
specifying work area into which an invitation list is to be moved 279
ID sequence of computer 20
identification sequence exchange 18, 24
invalid sequence 19
indicated on message error record 371
idle characters
inserting in message 169
example 178
idle line 103
IDLE operand 103
IEBUPDTE
using to make modules resident 373
example 373
IEDQXA 365
IEDQXC 353
sample JCL for invoking 353
specifying PARM = parameters for printed output 354
IFCEREPO system utility program for gaining access to error records 329
inactive entries displayed for a line 311
INACTVTD operator command 311
INBUF macro 146
specifying conditional execution 146, 147
inbuffer subgroup 112
functions of 112
identifying beginning of 146
identifying to handle incoming buffers 146
translating to EBCDIC 152
incident checkpoint records 334
automatic environment record when full 333
example using 336-338
operator commands causing 326
specifying 151
use 334
incoming group 111
required delimiter macro 145, 147
subgroups of 111
translating to EBCDIC 152
incoming message
checking buffers for identical characters 155
checking input sequence number 196
counting messages for origin station 154
counting segments for origin station 154
editing 168-179
identifying end of MH processing 147
identifying subgroup to handle buffers 146
loss indicated on message error record 370
maintaining record of traffic 167
removing EOA character 143
sample format 111
translating 137-139
INEND macro 147
INHDR macro 145
specifying conditional execution of 146
inheader subgroups 111
functions of 112
identifying beginning of 145
specifying execution of 146
translating to EBCDIC 152
initializing and activating the MCP 90-105
obtaining disk efficiency 75
INITIATE macro 163
effects on EOB checking 143
example 165
restrictions 164
specifying conditional execution 164
specifying use of EBCDIC blank characters 164
inmessage subgroups 112
functions of 112
identifying beginning of 147
required delimiter macro 147
INMSG macro 147
INOUT operand 102
input data control block 236
input data set 236
specifying 102
specifying DD statement 245
specifying format and characteristics of work unit 240
input DCB macro 238-241
input/output block
sense byte in operator awareness message 329
status bytes in operator awareness message 329
INPUT operand 102
input sequence number
checking 196
displaying for last message from a station 322
format of header field 196
inquiry/rapid response 286-289
coding considerations 288
sample program 213-219
inquiry-response application use of origin field in work area 255
insert operation 169
integer operand
CUTOFF macro 155
MSGLIMIT macro 183
ORIGIN macro 186
PRIORITY macro 190
SETSCAN macro 200
intensive-mode error recording 328
displaying current status of 322
specifying type 309
intercepted stations
destination queue restriction (reusable disk) 78
displaying list of 312
indicated on message error record 371
restrictions on holding messages 140
specifying 162
specifying another destination to receive messages queued for 285

use of HOLD macro 139
intercept function 168
queuing requirement 80
releasing messages 318
interface control check indicated on message error record 371
interface for MCP/application program 235-236, 41
defining components 236-249
internal code (EBCDIC) 387
interruption
I/O error recovery procedures 326
specifying number of entries in I/O trace table for a line 99
specifying point in routine to gain control when I/O trace table full 99
interval
automatic environment checkpoint record following 333
example 337
between computer-initiated calls to a switched station 33
between inquiry and response 286
specifying as additional feature 101
system 53
activating 311
changing duration of 322
specifying length 97
INTERVAL operator command 311
intervention required
indicated on message error record 372
specifying intensive-mode error recording for 328
INTRCEPT operator command 312
INTRO macro 92-101
checkpoint/restart operands 331
format 94
initialization and activation 91
list of functions 92
providing warning when main-storage queues nearly full 96
saving registers when user code handles multiple-buffer headers 97
specifying additional features 101
specifying COMWRITE output data set 100
specifying duration of mark character 100
specifying identifier for operator commands 95
specifying length of system interval 97
specifying maximum command input blocks used simultaneously 94
specifying maximum destination queues used simultaneously for application programs using CKREQ macros 98
specifying maximum simultaneous on-line tests 100
specifying maximum time between environment checkpoints 97
specifying maximum units simultaneously used for main-storage queuing 96
specifying message IEA001 to be displayed 100
specifying name of dead-letter queue 97
specifying name of MCP 94
specifying number of channel program blocks 94
specifying number of checkpoint request records 98
specifying number of entries for line trace 99
specifying number of entries in cross-reference table 99
specifying number of entries in STCB trace table 100
specifying number of environment records kept in checkpoint data set 97
specifying number of units for segments 96
specifying password for application program macros 99
specifying point in routine to gain control when line trace table full 99
specifying primary operator control station 95
specifying size of buffer unit 95
specifying type of lines 101
specifying type of restart 98
specifying when user informed that message queues no longer crowded 96
specifying whether message queues data sets are on a DASD 94
testing return code 101
INTVAL = operand 97
INTVL = operand
HOLD macro 163
line group DCB macro 67
invalid destination causing user-written routine to gain control 161
invitation 18
errors indicated on message error record 371
lines to multipoint BSC stations 20
specifying delay 67
invitation characters 19
invitation list
activating terminal entry of nonswitched station for entering messages 308
constructing 19-24
deactivating an entry in 313
delimiter 279
displaying active entries in 300
displaying inactive entries in 311
displaying status of polling for a line 320
dummy 68
entry (example) 20
modifying 282-284
specifying change or type of change for 283
recording status of 333
sample format 277-279
specifying entries for a line 19
example 20
specifying names of 68
use of relative line number in specifying entries 68
invitation message for audio terminals 20

INVLIST macro 19-24
examples 21-24
contention lines to terminals not assigned ID sequences 23
nonswitched lines to stations using polling characters 21
output-only lines to stations having no ID sequences 24
switched lines to stations using ID sequences 22
switched lines to terminals using polling characters 22
INVLIST = operand 68
Inward WATS (specifying telephone number) 32
irrecoverable error 326
terminating connection with station 326
ITB control character 17
retaining in incoming messages 143
specifying for outgoing messages 180

KEYLEN = operand 95

LAST = operand 26
LC = operand 143
leased line 15
length operand
READ macro 267
WRITE macro 268
LEVEL = operand
TERMINAL macro 32
TPROCESS macro 43
line address characters for IBM 2260 194
line address in operator awareness message 329
line code
AT&T 83B3 399
AT&T TWX 400
IBM 1030 390
IBM 1050 391
IBM 1053 printer 393-394
IBM 1060 392
IBM 2740 395
IBM 2741 (Correspondence) 398
IBM 2741 (EBCD) 397
IBM 2741 (hexadecimal equivalents of BCD) 396
IBM World Trade Telegraph ITA2 401
IBM World Trade Telegraph ZSC3 402
specifying 152
translation to EBCDIC 151
WU 115A 399
line

address insertion (sample user code) 196
arrangement when using WATS 52
coding the TERMINAL macro 37-40
contention 15
dummy invitation list (output only) 68
error indicated on message error record 371
idle 103
non-contention 15
nonswitched 15
stopping transmission for a station 315
point-to-point 15
recording changes in status 334
recording status of 333
specifying activation for line group data set 103
specifying intensive-mode error recording for 328
specifying transmission priority 67
specifying types for TCAM 101
starting or resuming transmission on 319
stopping transmission 321
line control 15-17, 233
defining 15
establishing 13
use of scan pointer during translation 152
line control block dump 344
line-control characters 16
in an application program 16
insertion 16
when to remove 17
line entry 25
defining in the terminal table 36
DSECT of 275
line group 433
changing polling delay of 317
characteristics of 65
data set 65-72
creating 65-72
examples of opening 103
guideline for specifying as input/output 103
restriction for coding as output 102
specifying activation of lines in 103
specifying DD statements 70-72
DCB macro 65-70
priority 48
specifying in order to modify invitation list 283
specifying to display invitation list for a line 279
starting or resuming transmission on 319
stopping transmission on 321
line I/O interrupt trace table 345
activating 305, 310
deactivating 314
examples of obtaining printed output 353
specifying number of entries 99
specifying point in routine to gain control when table full 99
line tone (specifying duration of mark character) 100
line trace 345
LINETYP= operand 101
linkage-editing an MCP (sample JCL) 206
list form specified on OPEN macro 103
LIST= operand 40
LNSTATUS operator command 312
LNUNITS= operand 96
LOCK macro 165
forms for coding 287
inquiry/rapid response coding considerations 288
restrictions 165
specifying conditional execution 166
specifying type of lock mode 166
specifying use of EBCDIC blank characters 166
lock mode 165, 286
extended 166
removing station from 202
message 166
response to inquiring station during quick closedown 287
specifying type 166
LOCOPT macro 167
specifying name of option field to be accessed 167
specifying register to contain address of option field 167
LOG macro 167
log data set 87
BSAM operands 87
data control block for 344
example of opening 103
specifying as output 102
specifying name of data control block 45
specifying where messages to be queued 45
LOG macro
logging messages or segments 344
restrictions 168
specifying location of log medium 168
logging
messages 168, 341-344
segments 168
types listed 168
logical errors
indicated on message error record 371
testing for 144
LOGICAL= operand 144
logtype entry 25
LOGTYPE macro 44
logging complete messages 344
specifying buffer size for messages destined for logging medium 45
specifying name of data control block 45
specifying where messages for logging medium to be queued 45
lost data
indicated on message error record 372
specifying intensive-mode error recording for 328
LRECL= operand
input DCB macro 240
output DCB macro 243
LTORG instruction 142

machine and device requirements 361-363
machine end-of-address (EOA) character 110
MACRF= operand
checkpoint DCB macro 86
input DCB macro 239
line group DCB macro 66
log DCB macro 87
message queues DCB macro 84
output DCB macro 243
macro formats
TCAM 367
macro instructions (see directory at front of book)
main-storage message queues data set
destination queue in 79
providing warning when nearly full 96
specifying backup
nonreusable disk 32
reusable disk 32
specifying maximum units used simultaneously 96
main-storage-only queues
lost message indication 370
specifying 32
main-storage queuing 79-81
specifying as additional feature 101
with disk backup 80
without disk backup 79
maintaining orderly message flow 45-53
mark character (specifying duration) 100
mask operand 116
CANCELMG macro 150
ERRORMSG macro 157
HOLD macro 162
MSGGEN macro 181
REDIRECT macro 193
MAXLEN= operand 26
MCP 13
ABEND formatted dump 344
activation and deactivation 91-107
activating and deactivating the application program interface 249-252
assembling 206
buffer definition checklist 59
buffer size 61
specifying for messages to MCP from application program 244

buffer unit pool 57-58
closedown 101
specifying type of restart following 98
coding requirements for message logging 342
coordinating restart with application program 293
data set initialization and activation 101
deactivation 105-107
TCAM with application programs 106
TCAM with no application program 105
effect of abnormal termination on application programs 293
execution 206
automatic environment checkpoint record 333
starting with catalogued procedures 207
functional MH macros 150-204
functions 13
initializing and activating 90-105
interface with application program 235-236, 41
defining components 236-249
line control 233
linkage-editing 206
priority 233
putting together 205-231
reconstructing for restart 331-341
specifying number of checkpoint request records 98
sample code 208-231
sections listed 205
specifying name 94
specifying size of buffers containing messages for application program 240
starting and restarting 91
terminal table 24
specifying logging complete messages 344
tracing flow of messages 341, 342
writing 13
user tasks in 13
MCPCLOSE macro 251, 252
restriction 289
specifying password 99, 252
specifying type of closedown 252
considerations for buffered terminals 50
message 109, 253
canceled 150
categorizing for processing 183
dynamically varying path of through an MH 187
format 109-111
incoming (see incoming message)
indicating EOF 198
input sequence number check 196
length error indicated on message error record 370
limiting number sent 182
loss avoided during warm restart 341
loss due to system failure 335, 336
marking serviced 335
operator awareness 329
outgoing (see outgoing messages)
output suspended to a station 162
parts of 109
processing as a work unit 259, 260
record 110
redirecting when unsent 192
reentering after system failure to prevent loss 336
releasing when intercepted 318
sample formats 111
selective translation 152
example 153
sequence number 196
displaying last from a station 322
displaying last to a station 322
specifying destination on FORWARD macro 160
testing for operator commands 152
example 153
text 109
translation 137-139, 152
avoiding 138
varying path within MH 129
warm restart after flush closedown to prevent loss 341
message and record processing 261
message block 109
message buffer format for TCAM/SAM SYNADAF 273
message editing 115, 168-179
message error record 369-372
bits described 369-372
displaying for a line 313
macros that set bits in 117
setting bits in 327
use of TERRSET macro with 202
message flow
example of 2-segment message with multiple-buffer header 126
example of 2-segment message with single-buffer header 125
logging 116
through an MH 122-128
to an application program 235
within MH group 123
message header 110-111
checking validity of origin field 185
controlling path of through an MH 183
delimiting destination field for multiple routing 161
destination codes in 110
format 110
date and/or time 155
input sequence number 196
locating fields in 132-135
scan pointer used for 118-121
using control characters to vary path of message in MH 129
message lock 286
message logging 341-344
coding requirements 342-344
complete messages 344
how it works 342
information flow 343
logging segments 344
selectivity 344
uses 341
what to log 342
message operand
LOCK macro 166
MSGGEN macro 181
message priority 45
after zone reorganization of DASD data set 77
efficient use of 78
queuing and 46-48
message processing 253
categorizing 183
guidelines for specifying 248, 249
Message Processing Program (QTAM) conversion 403
message queue
application program
recording status 334,335
specifying where maintained 42
main storage
providing warning when nearly full 96
specifying maximum units used simultaneously 96
main-storage-only lost message indication 370
recording status of 333
scanning 335-338
message queues data set 72-84
DCB macro for 83-84
DD statement for 84
disk
example of assigning relative record numbers 73
preformatting 365
DCB macro for 83-84
destination queue in main storage 79
destination queues on disk 73
disk efficiency 75
dump 353
main storage 79-81
opening (example) 103
preformatting DASD 75
reusable DASD 76-79
automatic environment checkpoint at zone changeover 333
designing for high message traffic 78-79
reorganizing 76
sample JCL for printed output 353
scanning queues in 335-338
specifying 83, 85

specifying DASD as input/output 102
specifying one or more 81-84
specifying PARM = parameters for printed output 354
specifying type in terminal table 31
specifying when user informed that message queues no longer crowded 96
specifying where maintained 31
specifying whether on a DASD 94
message retrieval facility 285, 286
POINT macro 285
message routing 116
control 13
techniques of coding for one or more destinations 159
message segments 55, 109, 123
expediting transmission of 163
maintaining incoming count for origin station 154
outgoing maintaining count for destination station 154
translation of 152
message subblock 109
message switching sample program 208-212
message transmission
between start-stop and BSC stations 16
dynamically varying path through an MH 187
establishing contact for 17
lost data indicated on message error record 370
specifying continuation after retry exhausted 144
specifying termination after retry exhausted 143
stopping for a nonswitched station 315
suspending 139
output 162
suspension to intercepted station indicated on message error record 371
MF = operand
application program 250
MCP 103
MFT-II 361
MH
base register requirements 145
conditional execution of functional macros 129
controlling path of message through 183
delimiter macros 140-149 (see directory at front of book)
designing 109-205
conditionally executing macros 129
delimiter macros 140-149
functional macros 150-204
hold/release facility 139
list of groups/subgroups/delimiter macros 141
message flow 122-128
message format 109-111
message header 110
message translation 137-139
order of macro specification 118
selecting functions 115-118
steps in 140
structure 111-114
user code in 130-137
variable processing in 129
dynamically varying path of message through 187
functional macros 150-204 (see directory at front of book)
functions provided 115
error handling 116
function modification 118
message editing 115
message routing 116
record keeping 116
system control 117
validity checking 116
gaining access to option fields 26
groups 112
delimiters 113
order of 113
incoming group 112
macro return codes 136
macros
delimiter 109
functional 109
order of specification 118
macros and corresponding subgroups listed 115
message flow 122-128
example of 2-segment message with multiple-buffer header 126
example of 2-segment message with single buffer header 125
message processed by application program 123
switched message 122
within group 123
minimum requirements 113
organization 112
outgoing group 112
purpose of 109
specifying address of for an application program 246
specifying for line group 69
subgroups and corresponding macros listed 115
subgroups
delimiters 113
functions of 112
order of 113
types of macros 109
user code in 130-137
closed subroutines 132
formula for determining bytes resulting from 131
locating header field 132-135
locating option fields 132
macro return codes 136
multiple-buffer header considerations 131
obtaining information from EXEC job-control statement 136
open subroutines 132
requirements and restrictions 131
sample activation of closed subroutine 133
variable processing in 129
varying path of message in 129
MH = operand
line group DCB macro 69
PCB macro 245
MINLIN = operand 418
modules associated with operator commands 374
MOVE = operand 169
MRELEASE macro 284-285
restriction 289
specifying password 99, 285
specifying station to receive intercepted messages 285
MSGEDIT macro 168-179
avoiding coding problem 172
EOB-checking restrictions 170
example of coding problem 171
examples 175-179, 194
deleting and replacing data 178
deleting data followed by contracting 177
deleting miscellaneous data 178
deleting several characters 179
deleting single character 179
inserting control symbols in segments 179
inserting data after every n bytes 177
inserting data in header buffer 176
inserting idle characters 178
inserting line addresses 194
multiple inserts and removes 179
replacing data 177
simultaneously inserting and replacing data 177
format 172
limitations 170
restrictions 170, 174, 175
scan pointer effects 171
specifying beginning of data to be removed 173
specifying data to be inserted 172, 173
specifying data to be removed 172, 173
specifying end of character string to be removed 175
specifying insert on remove operation 172
specifying location at which data to be inserted 173
specifying type of function 172
speed of execution 170
structure of operand groups 172
MSGFORM macro 179
specifying outgoing blocking factors 180
specifying outgoing ITB characters 180
specifying whether transparent mode used 180
MSGGEN macro 180
restrictions 182
specifying conditional execution 181

specifying data for 181
specifying logical connection between mask and message error record 181
specifying type of translation 182
MSGLIMIT macro 182
restrictions 182
specifying number of messages for a transmission sequence 182
MSGTYPE macro 183
example 184
specifying path of message through an MH 184
specifying use of EBCDIC blank characters 184
MSMAX= operand 96
MSMIN= operand 96
MSUNITS= operand 96
multiple-buffer header 123
considerations for user code in an MH 131
handling 124
multiple destinations 159
canceling messages to 150
multiple disk arms 75-76
multiple READ macros 266
multiple-subgroup restrictions 124, 127
multiple-wait capability 270
example 271
multiplexer channel 361
multipoint 15
multiprocessing 361
MVT 361

NCP= operand 87
network control facilities 273-285
network control macros 273, 274
ICHNG macro 282-284
ICOPY macro 276-279
MRELEASE macro 284-285
QCOPY macro 280-281
TCHNG macro 281-282
TCOPY macro 274-276
network reconfiguration 329-330
application program macro instructions for 330
operator commands for 329
NOENTRNG operator command 313
non-contention line 15
NONE operand 153
nonreusable disk queues 79
specifying 32
message retrieval from destination queue 285
preformatting 365
specifying for message queues data set 83
specifying percentage of records to be used before flush closedown 84
nonswitched line 15
activating station on 300
activating station's terminal entry for entering on 308
preventing station transmission to CPU 313, 315
nontransparent mode 17
specifying 180
NOTRACE operator command 314
NOTRAFIC operator command 315
NTBLKSZ= operand 34
nucleus generation 364

OBR extension for TCAM 327-329
OLT (see on-line test)
OLTEST= operand 100
on-line test 356-359
advantages 356
devices supported 356
invalid request indicated on message error record 370
specifying maximum that may occur simultaneously 100
system requirements 357-359
coding 358
JCL for TOTE/OLTs 358
main-storage 357
OS/SYSGEN 358
TOTE 357, 358
tests 356
TOTE facilities 356

OPDATA= operand
TERMINAL macro 34
TPROCESS macro 43
OPEN macro 101-104, 249-250
checkpoint/restart operands to be considered 332
examples of opening data sets 103
initialization and activation 91
specifying activation of line for line group data set 103
specifying list and execute forms 103
specifying name of data control block 250
specifying name of DCB macro 102
specifying type of data set 102
Open routine error codes returned by TCAM 88-89
operand formats 367
operating system generation 364
operator awareness message 329
for I/O error 326
specifying display of IEA001 100
operator commands 300-324 (see directory at front of book)
entering from application program 299
examples 297
incident checkpoints resulting from 326
incident records caused by 334
incorrectly formatted 299
listed by areas affected 325
listed by type of operation 405
operation types 296
queuing responses to 298
responses 298
specifying 297
specifying identifier for 95
operator control 295-325
activating debugging aids 305
activating line trace 310
activating nonswitched station for transmission 300-301
activating nonswitched station's entry for entering 308
activating the system interval 311
changing duration of polling delay 317
changing duration of system interval 324
checkpointing commands 326
command format 295
commands listed by operation 405
deactivating line trace 314-315
displaying active invitation list entries 300
displaying current status of intensive-mode recording 322
displaying list of inactive entries for a line 311
displaying list of intercepted stations 312
displaying message error record for a line 312, 313
displaying name of primary station 307
displaying names of secondary stations 307
displaying polling status of a line 320
displaying queue control block fields 317
displaying sequence number of last message to/from a station 322
displaying station's option fields 316
displaying station status 322
displaying station's relative line number 319
displaying status field for a line 312, 313
entering commands from application program 299
establishing primary station 303
incident records caused by commands 334
incorrectly formatted commands 299
initialization for 295
initiating closedown 324
inserting data in option fields 304
intensive-mode error recording 309, 328
making resident 373-374
operator commands (see directory at front of book)
preventing nonswitched station entering to CPU 313
preventing transmission for nonswitched station 315
primary station
displaying name of 307
operator awareness message 329

queuing responses to commands 298
releasing intercepted messages 318
replies to a component 36
responses to commands 298, 299

sample commands 297
secondary stations
displaying names of 307
specifying an application program in the terminal table 42
specifying commands 297
specifying primary station 95
specifying secondary stations in the terminal table 35
specifying temporary I/O error records 308
starting or resuming line transmission 319
stopping transmission for line or line group 321
suspending transmission to a station 323
switching to Auto Poll 303
switching to programmed polling 301
testing for operator commands 152
example 153
opfield operand
COUNTER macro 154
LOCOPT macro 167
MSGLIMIT macro 182
PATH macro 187
OPTCD= operand
checkpoint DCB macro 86
input DCB macro 240
message queues DCB macro 84
output DCB macro 243
OPTFIELD operator command 316
option fields 434
automatic recording of changes at cold restart 334
defining in the terminal table 26
displaying for a station 316
examining contents 274
gaining access to
specifying name for 167
specifying register to contain address for 167
inserting data in 304
locating 132
macros that may gain access to 26
modifying 281-282
moving contents to work area 275
recorded on environment checkpoint record 333
recording changes in status 334
recording status 334
reserving space in 26
specifying actual data to be inserted 34, 37, 43
example 37, 44
specifying incident checkpoint record of 151
specifying type and length in terminal table 27
specifying work area to contain 276
option table 26
displaying fields in 316
specifying address of field in 167
OPTION macro 26-29
coding examples 28-29
specifying type and length of option field 27
optional fields
defining in work area 254-257
origin and destination 255
position field 255
SAM prefix 256
format of relative positions in work area 257
included in specifying length of work unit
READ macro 267
WRITE macro 268
included in specifying size of work area 239
included in specifying record size in work area 240
included in specifying work-unit size in work area 243
specifying for work unit 240, 243
ORDER= operand 19
ORIGIN macro 185
origin code error indicated in message error record 369
origin field
in work area 254
message header validation 185, 187
relative position among optional fields in work area 257
ORIGIN macro
specifying character count for origin fields in a message header 186
variable functions of 185
origin station
maintaining incoming count of messages 154
of message segments 154
maintaining count of outgoing message segments for 154
OS generation 364
outboard recorder extension for TCAM 327-329
OUTBUF macro 148
specifying conditional execution 149
outbuffer subgroups 112
functions of 112
identifying to handle outgoing buffer 148
translating to line code 152
OUTEND macro 149
outgoing group 112
required delimiter macro 149
subgroups of 112
outgoing message
counting segments for destination station 154
editing 168-179
identifying subgroup to handle buffers 148
inserting sequence number 196
maintaining count for destination station 154
maintaining record of traffic 167
sample format 111
specifying blocking factors 179
specifying priority handling for 190
translating 137-139
outgoing subgroup translation to line code 152
OUTHDR macro 148
specifying conditional execution 148
outheader subgroup 112
functions of 112
identifying beginning of 148
translating to line code 152
outmessage subgroup 112
functions of 113
required delimiter macro 149
OUTMSG macro 149
specifying conditional execution 149
output data control block 236
output data set 236
COMWRITE requirements and format 347
DCB macro 242-245
specifying DD statement 245
specifying format and characteristics of work unit 244
specifying type on OPEN macro 102
output DCB macro 242-245
OUTPUT operand 102
output sequence number
displaying last for a station 322
inserting 196
Outward WATS (interval between computer-initiated calls to switched stations) 33
overrun
indicated on message error record 372
specifying intensive-mode error recording for 328

parameter list on OPEN macro 250
parity character error indicated on message error record 372
password for application program macros 99
PASSWRD= operand
ICHNG macro 284
INTRO macro 99
MCPCLOSE macro 252
MRELEASE macro 284
TCHNG macro 282
path switches 129
altering to vary path of message through an MH 187
specifying execution of inheader subgroup 145
path switching delimiter macros 118
PATH macro 187-189
example 188
specifying conditional execution 188
specifying path switch setting 187
specifying path-switch byte to be used 187
specifying use of EBCDIC blank characters 188
PATH= operand
INBUF macro 146

INHDR macro 146
INMSG macro 147
OUTBUF macro 148
OUTHDR macro 148
OUTMSG macro 149
PCB macro 245-246
specifying bytes for date/time/sequence number 246
specifying initial buffers to handle data in user work area 246
specifying initial buffers to handle data obtained by GET/READ 246
specifying MH for an application program 246
specifying size of buffers for an application program 246
PCB = operand 42
PCI = operand 69
permanent-error record 326, 327
access to 329
point-to-point line 15
POINT macro 285
restriction 289
specifying data control block for message retrieval 286
specifying required address of a field 286
POINT = operand 201
POLLDLAY operator command 317
polling 18
changing duration of delay 317
determining type for a line 278
displaying status of a line with respect to 320
timeout exceeded indicated on message error record 372
polling characters 18
for polled stations 19
position field 255
guidelines for using in work area 248
relative position among optional fields in work area 257
prefix 55
primary operator control station 434
command for establishing 303
displaying name of 307
operator awareness message 329
specifying 95
PRIMARY = operand 95
printing utility
COMEDIT 352
examples of invoking 353
PARM = options 352
IEDQXC (PARM = options) 354
priority
application program 233
equal
nonswitched contention stations 50
nonswitched polled station with programmed polling 49
nonswitched polled stations with Auto Poll 49
MCP 233
message 45-48
after zone reorganization of DASD data set 77
busy lines 52
efficient use of 78
receive
nonswitched polled stations with Auto Poll 49
nonswitched polled stations with programmed polling 48
send
nonswitched contention stations 50
nonswitched polled stations with Auto Poll 49
nonswitched polled stations with programmed polling 49
specifying 32
permissible levels for messages on a process queue 43
transmission 45, 48-51
efficiency when receive specified 78
specifying for line 67
priority handling for outgoing messages 190
PRIORITY macro 190
effect of CODE macro on 190
example 192
specifying conditional execution 190
specifying for a message 190
specifying priority level for a message 190
specifying use of EBCDIC blank characters 191
private library definition 207
problem program exit list
specifying address on checkpoint DCB macro 86
specifying address on message queues DCB macro 84
process control block
defining in the MCP 237-246
dump of 344
specifying name of in the terminal table 42
PCB macro for 245-246
process entry 25
application program requirements 236
process queue 434
specifying priority levels of messages on 43
processor/library generation 364
PROGID = operand 94
program-controlled interruption
buffer allocation considerations 62
specifying for buffer allocation/deallocation 69
program EOA (example) 110
programmed polling 48
determining use on a line 278
switching to 301
protection password for application program macros 99
purging destination queue at restart 42
PUT macro 265
specifying address of data control block 265
specifying address of work area 265
specifying to gain access to checkpoint data set 86
specifying to gain access to line group data set 66
specifying to gain access to message queues data set 84

QBY = operand 30
QCOPY macro 280-281
restriction 289
specifying name of terminal table entry whose queue control block is to be displayed 280
specifying name of work area for displaying queue control block 281
QSTART macro 290
in checklist for checkpoint/restart 332
QSTATUS operator command 317
QTAM
converting application programs 403
macro facilities listed 403
queue 434
queue control block
displaying fields in 317
dump of 344
examining 280-281
master 280
specifying name of terminal table entry for displaying 280
priority 280
QUEUES = operand
LOGTYPE macro 45
TERMINAL macro 31
TPROCESS macro 42
queuing and message priority 46-48
queuing
checkpoint facility requirements 80
disk 72-79
advantages and disavantages 73-74
intercept function requirement on 80
main storage 79-81
with disk backup 80
without disk backup 79
messages for one or more destinations 159-161
retrieve function requirement on 80
specifying one or more methods 81-84
specifying main-storage as additional feature 101
specifying reusable disk as additional feature 101
techniques listed 72
queuing by destination 72
queuing by line
considerations for 46
example 47-48
queuing by terminal

considerations for 46, 47
example 47-48
quick closedown 101
automatic environment checkpoint record during 333
cold restart following 340
initiating through operator control 324
response to inquiring station in lock mode 287
specifying in application program 252
warm restart following 340
QUICK operand 252

read-ahead queue 235, 247
formula for number of buffers required for 248
role in message flow 122
use of 246
READ macro 265
issuing more than one per process queue 266
specifying address of data control block 267
specifying address of work area 267
specifying initial buffers to handle data obtained by 246
specifying length of work unit plus optional fields 267
specifying name of data event control block 267
specifying SF 267
READY macro 104-105
initialization and activation 91
RECDEL = operand 43
receiving 15
BSC considerations 414
RECFM = operand
input DCB macro 240
log DCB macro 87
output DCB macro 244
TPEDIT macro 418
record 110, 253
checkpoint request 334
priority used in dividing space for 339
specifying data set for 86
specifying number for checkpoint data set 98
control 333
counter overflow 328
delimiting for an application program 43, 258
end-of-day 328
environment checkpoint 333
example using 336-338
how updated 334
error
displaying current status of intensive-mode recording 322
gaining access to 329
intensive mode recording 328
I/O error types 327-328
incident checkpoint 334
automatic environment record when full 333
example using 336-338
operator commands causing 326
specifying 151
use 334
inserting delimiter (example) 178
maintaining for message traffic 167
permanent-error 326
processing as a work unit 260-263
specifying size 240
types in checkpoint data set 333-335
record keeping 116
record and message processing 261
record processing guidelines 248-249
recoverable error
operator awareness message indicating failure 329
retrying the block 327
REDIRECT macro 192
specifying additional destinations 193
specifying conditional execution 193
specifying connection between mask and message error record 193
(*register*) operand
CODE macro 153
LOCOPT macro 167
registers
saving when user code handles multiple-buffer headers 97
specifying for an MH 145
relative line number 434
CPU calling a station 52
displaying for a station 319
example of TERMINAL macros arranged according to 39
specifying 31
to display invitation list 279
to modify an invitation list 283
station calling the CPU 51
use in invitation list 68
releasing intercepted messages 318
RELEASE operand 163
remove operation 169
REPLACE = operand 421
RESERVE = operand
line group DCB macro 69
PCB macro 246
RESMXMIT operator command 318
response (see inquiry/rapid response)
response keywords at INTRO execution time 93
restart 330-341
building "Good Morning" and "Restart in Progress" messages 104-105
checkpoint data set for 85-86
cold 340
after abnormal flush closedown 79
automatic recording of changes in option fields 334
building "Good Morning" message 104-105
good morning routine gaining control following 340
reformatting DASD message queues data sets 75
conditions required for various types 341
continuation 340
replaced by cold restart due to faulty checkpoint records 340
coordinating MCP and application program 293
maintaining continuity of sequence numbers 198
purging destination queue 42
scanning message queues after system failure 335-337
types 339-341
specifying 98
warm 340
replaced by cold restart due to faulty checkpoint records 340
restart in progress routine (use) 336
restarting the MCP 91
RESTART = operand 98
RESULT = operand 201
retrieval of messages 285-286
POINT macro 285
queuing requirement 80, 81
retry count exhausted
continuing message transmission 144
terminating message transmission 143
retrying the block 327
reusable disk queuing 76-78
advantage of combining checkpoint coordination methods for 293
automatic environment checkpoint at zone changeover 333
designing for high message traffic 78
preformatting 365
retrieving messages from destination queue 285
specifying 32, 101
for message queues data set 83
rln operand
ICHNG macro 283
ICOPY macro 279
RLN = operand 31
RLNSTATN operator command 319
RSMSG = operand 105

SAM prefix 256-257
SAM/TCAM compatibility 289-293
sample programs 208-231
file updating with checkpoint coordination 220-231
inquiry/rapid response 213-219

message switching 208-212
scan 335-338
scan pointer 118-121
automatically moving 118
coding considerations 120
error indication on message error record 369
example of use 119
MH macros not dependent upon 124
moving 199-202
use of with CODE macro 152
SCREEN macro 194-196
example 196
return codes 195
specifying conditional execution 195
specifying type of Write operation 195
specifying use of EBCDIC blank characters 195
use of MSGEDIT macro with 194
SCT= operand 70
SDR extension for TCAM 327-329
secondary operator control station
displaying names of 307
specifying an application program as 42
specifying in the terminal table 35
secondary storage for message queues data sets 94
SECTERM= operand
TERMINAL macro 35
TPROCESS macro 42
segments 109
expediting message transmission using INITIATE macro 163
incoming
maintaining count for origin station 154
maintaing record of 167
outgoing
maintaining count for destination station 154
maintaining record of 167
seizing the line 15, 24
selection 18, 24
errors indicated on message error record 371
sending 15
BSC considerations 414
SENDTRP= operand 180
sense byte in input/output block of operator awareness message 329
sense count 323
SEQUENCE macro 196
reserving bytes for sequence number 246
sequence number
displaying for last message to or from a station 322
errors indicated on message error record 369-370
input 432
checking 196
format of header field 196
internal counter for 197
maintaining continuity during restart 198
output 434
inserting 196
recording status of fields containing 334-335
reserving bytes in buffer for 69, 246
service bit 335
service facilities 295-359
checkpoint/restart 330-341
I/O-error recording 327-329
I/O error-recovery procedures 326
message logging 341-343
network reconfiguration 329-330
operator control 295-325
SETEOF macro 198
specifying conditional execution 198
specifying use of EBCDIC blank characters 198
SETSCAN macro 199-202
examples 202
format 200
specifying direction of scan pointer movement 201
specifying new location of scan pointer 200
specifying register to contain address of last character 201
specifying use of EBCIDIC blank characters 200
specifying whether scan pointer to remain stationary after a move 201
SF operand
READ macro 267
WRITE macro 268
single entry 24
DSECT of 274-275
SIO command counter in terminal-table entry 327
skipchars operand 200
special characters table name (specifying) 70
START command 207
Start I/O command counter in terminal-table entry 327
start-stop transmission 16
device-dependent considerations 407-413
starting and restarting the MCP 91
building "Good Morning" message 104-105
STARTLINE operator command 319
STARTMH macro 142-145
format 143
function of 112
inquiry/rapid response coding considerations 288
removing line control characters 143
retaining line control characters 143
specifying continuation of transmission after retry exhausted 144
specifying EOB completion handling 144
specifying number of base registers 145
specifying termination of transmission after retry exhausted 143
specifying tete-a-tete interaction 144
testing for logical errors 144
STARTUP= operand 98
STATDISP operator command 320
station 15
defining a component belonging to 25
defining in a group 24
defining individually 24
designating to receive user-specified error messages 157
displaying input sequence number of last message from 322
displaying option fields 316
displaying output sequence number of last message to a station 322
displaying relative line number of 319
displaying status of 322
error indicated on message error record 371
intercepted
indicated in operator control display list 312
indicated on message error record 371
restriction on type of destination queue 78
restrictions on holding messages 140
specifying 162
specifying another station to receive messages queued for 285
use of HOLD macro 139
invalid identification sequence indicated on message error record 371
limiting number of messages sent to 182
locking to application program 165
maintaining count of outgoing messages for 154
nonswitched
activating 300
activating terminal entry for entering 308
preventing transmission to CPU 313
stopping transmission for 315
operator control
command for establishing 303
specifying secondary in the terminal table 35
origin
maintaining count of incoming message segments for 154
maintaining count of incoming messages for 154
maintaining count of outgoing message segments for 154
primary operator control
displaying name of 307
operator awareness message sent to 329
removing from extended lock mode 202
secondary operator control
displaying names of 307
specifying an application program 42
specifying incident checkpoint records of option fields belonging to 151
specifying intensive-mode error recording for 328

suspending transmission to 139, 323
station control block dump 344
statistical data recorder extension for TCAM 327-329
statname operand
MRELEASE macro 285
TCOPY macro 276
status bytes in operator awareness message 329
status field displayed for a line 312
status information for debugging 305
STCB trace table 347
activating trace 305
examples of obtaining printed output 353
specifying number of entries for table 100
STOP= operand 143
STOPLINE operator command 321
considerations for buffered terminals 50
STOPLN macro (QTAM) 403
considerations for buffered terminals 50
STSTATUS operator command 322
STX control character 16, 17
removing from incoming messages 143
SUBBLCK= operand 180
subblock 109
size 34, 37
subgroups 111
arrangement 113
executing by setting switch 129
functions of 112
inbuffer
identifying beginning of 146
identifying to handle incoming buffers 146
translating to EBCDIC 152
inheader
identifying beginning of 145
specifying execution of 146
translating to EBCDIC 152
inmessage
identifying beginning of 147
required delimiter macro 147
outbuffer
identifying to handle outgoing buffer 148
translating to line code 152
outgoing
translating to line code 152
outheader
identifying beginning of 148
translating to line code 152
outmessage
required delimiter macro 149
restrictions on multiple 127
subtasks
keeping record of activation 347
attached 365
optional
attaching checkpoint 330-341
attaching COMWRITE 346-347
attaching on-line test 356-358
required
attaching operator control 295-300
SUSPXMIT operator command 323
switched line 15
defining for input or input/output 25
identifying station for incoming calls 185
use of TERMINAL macro 37, 38
switches 129
switch operand 187
SYNAD
exit 271
input to routine 272
register contents on entry to 272
status indicators of routine 272
SYNAD address
specifying on input DCB macro 241
specifying on output DCB macro 244
SYNADAF 271,273
format of TCAM/SAM message buffer 273
SYNAD= operand
input DCB macro 241
output DCB macro 243
SYNADRLS macro 271
SYS1.LINKLIB (making transient modules resident) 373-374
SYS1.LOGREC (gaining access to error records on) 329
SYSCLOSE operator command 324
considerations for buffered terminals 50
SYSGEN 364
SYSINTVL operator command 53, 324
system control 117
system failure
cold restart following 340
continuation restart following 340
restarting from 330
scanning message queues during restart 335
example 337
specifying type of restart for 98
suggestions for establishing checkpoint coordination 291-292
system generation considerations 364
system interval 53
activating 311
changing duration of 324
specifying length 97
system macros issued in an MH 131
system preparation 361-366
system records of changes in status 334

tablename operand 153
TBLKSZ= operand 34
TCAM/SAM compatibility 289-293
use of SAM prefix 256
TCAM
closing system 251-252
determining presence in CPU from non-buffered terminal 278
machine and device requirements 361-364
macro formats 367
conventions used 367
making transient modules resident 373-374
modules associated with operator commands 374
multiprocessing 364
running QTAM application programs 403
service facilities 295-359
specifying during system generation 364
system preparation 361-366
TCHNG macro 281-282
restriction 289
specifying name of work area containing replacement for terminal table entry 282
specifying password 99, 282
specifying terminal table entry whose contents are to be replaced 281
TCOPY macro 274
restriction 289
specifying station whose terminal-table contents are to be moved 276
specifying work area into which terminal table contents are to be moved 276
telecommunications system
macros used for controlling and modifying 117
specifying line configuration and device requirements 364
telephone number of a station (specifying) 32
teleprocessing network indentification 13
temporary error 327
counter in terminal-table entry 327
temporary-error record 328
access to 329
intensive-mode recording for 328
TERM= operand 31
terminal entry 24
activating nonswitched station for entering 308
types 24, 25
TERMINAL macro 29-36
abnormal termination due to improperly specified message queues data set 32
addressing characters for specifying a component 36
coding for a component 36-37
coding for a line 37
defining a component 36, 37
defining a line entry 36
format 30

multiple macros arranged by relative line number (example) 39
operands 30-36
specifying a component 36
options for specifying terminal type 31
overriding block size with the MSGFORM macro 34
relevant operands when specified for a line 39-40
specifying addressing characters 32
specifying alternate destination 33
component 36
specifying block and subblock sizes for a component accepting messages in nontransparent mode 37
specifying block size for outgoing messages
nontransparent mode 34
transparent mode 34
specifying block size of messages in transparent mode to a component 37
specifying buffer size for outgoing messages 33
specifying data control block name for line group 31
specifying data for option fields 34
example 35
specifying delay between message blocks sent to a buffered station 33
specifying interval between computer-initiated calls to a switched station 33
specifying priorities 32
specifying relative line number 31
specifying secondary operator control stations 35
specifying subblock size for outgoing messages in nontransparent mode 34
specifying telephone number of a station 32
specifying time for computer-initiated calls 33
specifying type of message queuing 31
specifying where message queues to be maintained 31
specifying whether a component is to accept replies to operator commands 36
summary determining use for a line 37-38
use of relative line number in dialing 31
using for audio lines 37
terminal
defining 15
determining number that are accepting 278
determining total number on a line 278
modifying Write operation 194
recording changes in status 334
recording status of 333
specifying type (available options) 31
types supported 361
terminal table 24
constructing 24-45
defining boundaries 25
defining option fields 26
device-characteristics fields in 275
DSECT format 275
dump 344
macro instructions
LOGTYPE 45
OPTION 26
TERMINAL 29-36
TLIST 40
TPROCESS 41-44
TTABLE 25
reserving space in an option field 26
specifying data for option fields (example) 44
specifying last entry in 26
specifying logging complete messages 344
specifying secondary operator control stations 35
types of terminal entry 24-25
terminal table entry
count of start I/O commands 327
count of temporary errors 327
examining contents 274
modifying 281-282
specifying name for displaying queue control block 280
specifying name of for replacing contents 281
termination
application program 293
due to not specifying user error-analysis routine 87
due to uncorrectable I/O error 87
improperly specified message queues data set 32
overlaying records on message queues data set 78
specifying user exit 87-90
wrapping nonreusable disk during flush closedown 79
terminology 15
termname operand
QCOPY macro 280
TCHNG macro 281
temporary I/O error
intensive-mode recordings 328
specifying records 308
TERRSET macro 202
setting a bit in the message error record 202
using ERRORMSG macro with 202
tete-a-tete interaction (specifying) 144
text 109
text buffer 55
text-only message 109
text transfer error indicated on message error record 371
THRESH = operand 84
time
format for inserting in header 156
logging 342
reserving bytes in buffer for 69, 246
specifying whether to be inserted in header 156
time-of-day for computer-limited calls 33
TIME = operand 156
timeout exceeded
indicated on message error record 372
specifying intensive-mode error recording for 328
TLIST macro 40
defining either distribution list or cascade list 40
example of extended list of entries 41
specifying entry for distribution list or cascade list 40
TO operand 175
TOPMSG = operand 100
TOTE 356
facilities 356
invalid request indicated on message error record 370
TP Op code in operator awareness message 329
TPEDIT macro 417-426
TPROCESS macro 41-44
checkpoint/restart operands 332
delimiting a record for the application program 43
interface between MCP and application program 41
operands 42-44
purging destination queue at restart 42
specifying actual data for option fields 43
example 44
specifying alternate destination 42
specifying application program as secondary station 42
specifying name of process control block 42
specifying permissible priority levels for messages on a process queue 43
specifying where application program message queues maintained 42
TRACE = operand 99
TRANS = operand 69
transferring data between MCP and application program 252-271
transient modules (making resident) 373-374
translation 16, 137-139
avoiding 138
of data in buffers 151
selective 153
specifying type 153
translation tables 138
formatting 138
list of TCAM-provided 138
overriding for a line group 152
providing 138
specifying 17
for line group 69
user 153
transmission 16
dynamically varying path of message through an MH 187
limiting number of messages to a destination 182
maintaining count of messages or message segments 154
specifying continuation after retry exhausted 144
starting or resuming on lines 319
stopping for a nonswitched station 315
stopping for line or line group 321

suspending 139
suspending output to a station 162, 323
transmission control unit 435
examining sense byte for I/O error 326
transmission priority 45, 48-51
efficiency when receive specified 78
nonswitched contention stations 50
nonswitched polled stations 48-49
Auto Poll 49
TCAM program poll 48-49
using buffering 49-50
specifying for line 67
switched stations 50-51
BSC 413
non-BSC 50-51
TWX 413
transparent mode 17
specifying for message transmission 180
TREXIT= operand 99
TTABLE macro 25
specifying last entry in terminal table 26
specifying length of terminal table name 26
typelength operand 27
typename operand 168
TYPE= operand 40

undefined error indicated on message error record 371
undefined-format work unit 257
input data set 240
output data set 244
undefined-length work unit size 243
undefined record delimiter 258
unit 55
allocation 57-58
examples 58, 59
deallocating from end of buffer 147
determining number needed 61
specifying maximum simultaneously used for main-storage queuing 96
specifying number for segments 96
specifying size for buffers 95, 96
unit exception
indicated on message error record 371
specifying intensive-mode error recording for 328
unit pool 55, 57-58
UNITSZ= operand 95
UNLOCK macro 202
examples 203
inquiry/rapid response coding considerations 288
specifying conditional execution 203
specifying use of EBCDIC blank characters 203
USASCII code 388
user code in MH 130-137
formula for determining bytes resulting from 131
USEREG= operand 97
user error indicated on message error record 371
UTERM= operand 36

variable-format work unit 257
input data set 240
output data set 244
variable processing within MH 123
variable record delimiter 258
VERCHK= operand 419
wait state
application program 269
during closedown procedures 251
WAIT macro for testing BSAM/TCAM completion codes 269
warm restart 340
replaced by cold restart due to faulty checkpoint records 340
WATS (arrangement of TCAM lines) 52
WDC operation 194, 195
WLA function 194
WLA operand 195
work area 235, 253
addressability requirements 131
contents described in position field 241, 244
defining 253-257
defining optional fields 254-257
origin and destination 255
position field 255
SAM prefix 256
dynamic definition of 254
format of relative positions of optional fields in 257
guidelines for using position field in 248-249
including optional fields when specifying size of 240,243
moving contents of option fields to 275
moving data between input and output 254
optional fields included in specifying record size 240
origin field in 255
position field in 255
format 241, 244
SAM prefix in 256
specifying address
GET macro 264
READ macro 267
specifying from which terminal table contents are to be moved 282
specifying initial buffers to handle data from 246
specifying into which invitation list contents are to be moved 279
specifying into which terminal table contents are to be moved 276
specifying on PUT macro 265
specifying size 239,243
specifying station whose terminal-table contents are to be moved to 276
static definition of 253
work unit 235, 252
effect of type and format on determining size of 263
formats 257-259
input DCB macro summary 258
output DCB macro summary 259
optional fields included in specifying work-unit size 243
processing
for a message 259
for a record 260-263
specifying 257-263
specifying input data set format and characteristics 240
specifying length including optional fields
READ macro 267
WRITE macro 268
specifying optional fields for 240, 243
specifying output data set format and characteristics 244
specifying size 243
types 259-263
WRE function 194
WRE operand 195
Write-at-Line-Address 194
WRITE macro 267
specifying address of data control block 268
specifying length of work unit plus optional fields 268
specifying name on data event control block 268
Write operation
modifying for terminals with display screens 194
specifying type for 2265s (Remote) or 2260s 195
verifying type in effect 195
WTOR at INTRO execution time 93, 94
WTTONE= operand 100
zone changeover (automatic environment checkpoint record) 333
example 337

READER'S COMMENT FORM

IBM System/360 Operating System
Telecommunications Access Method (TCAM)
Programmer's Guide and
Reference Manual

Order No. GC30-2024-0

- How did you use this publication?

  As a reference source ☐
  As a classroom text ☐
  As a self-study text ☐

- Based on your own experience, rate this publication . . .

As a reference source:	Very Good	Good	Fair	Poor	Very Poor
As a text:	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation?

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U. S. A.

YOUR COMMENTS, PLEASE . . .

This publication is one of a series that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, help us produce better publications for your use. Each reply is carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in using your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 569
RESEARCH TRIANGLE PARK
NORTH CAROLINA

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
P. O. Box 12275
Research Triangle Park
North Carolina 27709

Attention: Publications Center, Dept. E01

Fold Fold

Cut Along Line

IBM System/360 OS TCAM PG (S360-30) Printed in U. S. A. GC30-2024-0

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

GC30-2024-0

IBM System/360 OS TCAM PG (S360-30) Printed in U. S. A. GC30-2024-0

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)