SH20-9003-2

Program Product

Customer Information Control System/Virtual Storage (CICS/VS) Application Programmer's Reference Manual

Program Numbers 5740-XX1 (CICS/OS/VS)
5746-XX3 (CICS/DOS/VS)

IBM

Page of SH20-9003-2
Revised May 22, 1975
By TNL SN20-9086

Third Edition (March 1975)

This edition, as amended by technical newsletter SN20-9086, applies to Version 1, Modification Level 1, Release 1, of the program product Customer Information Control System/Virtual Storage (CICS/VS), program numbers 5740-XX1 and 5746-XX3, and to all subsequent versions and modifications until otherwise indicated in new editions or technical newsletters.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using this publication, consult the latest *IBM System/360 and System/370 Bibliography,* GA22-6822, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

A form for readers' comments has been provided at the back of this publication. If the form has been removed, address comments to IBM United Kingdom Laboratories Ltd., Publications Department, Hursley Park, Winchester, Hampshire, SO21 2JN, England. Comments become the property of IBM.

PREFACE

This publication contains detailed information necessary to design and prepare application programs to execute under either of two IBM program products: CICS/DOS/VS (5746-XX3) and CICS/OS/VS (5740-XX1). It is intended for use by application programmers, system programmers, system analysts, and system administrators. The manual contains both tutorial and reference material, as follows:

The first two chapters are tutorial in nature. They explain the general approach to application programming under CICS/VS. The next three chapters provide reference information concerning use of symbolic storage definitions in Assembler-language, American National Standard (ANS) COBOL, and PL/I application programs.

Chapter 6 is a tutorial explanation of the data management and supervisory services provided by CICS/VS. The CICS/VS macro instructions by which these services are requested are briefly identified. Chapter 7 contains reference information--the detailed formats of these CICS/VS macro instructions.

Chapter 8 provides tutorial and reference information concerning system monitoring services available under CICS/VS. The format established for reference material in the preceding chapter is maintained for reference material in this chapter. Chapter 9 begins with a tutorial introduction to built-in functions provided within CICS/VS and concludes with explanations of the detailed formats of macro instructions used to invoke these functions.

A tutorial discussion of application programming considerations (Chapter 11) is followed by a chapter containing both tutorial and reference information for programmers utilizing DL/I services. The appendices contain reference summaries of CICS/VS control areas, mnemonics, macro instructions, and translate tables, and Assembler-language, ANS COBOL, and PL/I sample programs.

For further information concerning CICS/VS and related subjects discussed in this manual, the reader is referred to the publications listed in the Bibliography. A glossary of certain terms applicable to CICS/VS is provided in the CICS/VS General Information Manual.

CONTENTS

CHAPTER 1. INTRODUCTION 1
Programming Techniques 6
Application Program Packing. 9
Quasi-Reentrance 10
CICS/VS Macro Instructions 10
  Coding Aids. 12
Storage Definition 12
Program Initialization 13
Restrictions 14
  American National Standard (ANS) COBOL 14
  PL/I 14
  Link-Editing 15
Considerations for a Virtual Storage Environment 15
Assembly-Time Service. 17

CHAPTER 2. STORAGE DEFINITION 19
Symbolic Storage Definitions 19
Required Storage Areas 24
  Common System Area (CSA) 24
    Common Work Area (CWA) 25
  Task Control Area (TCA). 25
    Transaction Work Area (TWA). 29

CHAPTER 3. COPYING STORAGE DEFINITIONS - ASSEMBLER LANGUAGE . . 31
Static Storage Definition. 31
  Common System Area (CSA) 31
  Terminal Control Table Terminal Entry (TCTTE). 31
Dynamic Storage Definition 32
  Task Control Area (TCA). 32
  Terminal Input/Output Area (TIOA). 33
  File Input/Output Area (FIOA). 33
  File Work Area (FWA) 34
  VSAM Work Area (VSWA). 34
  Transient Data Input Area (TDIA) 35
  Transient Data Output Area (TDOA). 35
  Temporary Storage Input/Output Area (TSIOA). 36
  Storage Accounting Area (SAA). 36
  Journal Control Area (JCA) 37
Example of CICS/VS Assembler-Language Application Program. . . . 37

CHAPTER 4. COPYING STORAGE DEFINITIONS - ANS COBOL. 39
Static Storage Definition. 40
  Common System Area (CSA) 40
  Terminal Control Table Terminal Entry (TCTTE). 40
Dynamic Storage Definition 40
  Task Control Area (TCA). 41
  Terminal Input/Output Area (TIOA). 41
  File Input/Output Area (FIOA). 42
  File Work Area (FWA) 42
  VSAM Work Area (VSWA). 43
  Transient Data Input Area (TDIA) 43
  Transient Data Output Area (TDOA). 43
  Temporary Storage Input/Output Area (TSIOA). 44
  Storage Accounting Area (SAA). 44
  Journal Control Area (JCA) 45
Additional Guidelines. 45
Example of CICS/VS ANS COBOL Application Program 48

CHAPTER 5. COPYING STORAGE DEFINITIONS - PL/I 51

Static Storage Definition. 51
Common System Area (CSA) 51
Terminal Control Table Terminal Entry (TCTTE). 52
Dynamic Storage Definition 52
Task Control Area (TCA). 52
Terminal Input/Output Area (TIOA). 52
File Input/Output Area (FIOA). 53
File Work Area (FWA) 54
VSAM Work Area (VSWA). 54
Transient Data Input Area (TDIA) 54
Transient Data Output Area (TDOA). 55
Temporary Storage Input/Output Area (TSIOA). 55
Storage Accounting Area (SAA). 56
Journal Control Area (JCA) 56
Example of CICS/VS PL/I Application Program. 57

CHAPTER 6. SERVICE INVOCATION 59
Terminal Services. 59
Write Data to a Terminal (WRITE) 62
Read Data from a Terminal (READ) 65
Synchronize Terminal Input/Output for a Transaction (WAIT) . . 66
Converse with a Terminal (CONVERSE). 66
Disconnect a Switched Line (DISCONNECT). 66
Examples 66
File Services. 67
Randomly Retrieve Data from a Data Set (GET) 69
Random Read-Only Retrieval 71
Random Retrieval for Update. 73
Random Retrieval through Indirect Access 74
Randomly Update or Add Data to a Data Set (PUT). 76
Randomly Delete Data from a Data Set (DELETE). 78
Obtain a File Work Area (GETAREA). 79
Release File Storage (RELEASE) 80
Browsing 82
Initiate Sequential Retrieval (SETL) 83
Retrieve Next Sequential Record (GETNEXT). 86
Terminate Sequential Retrieval (ESETL) 90
Reset Sequential Retrieval (RESETL). 91
Test Response to a Request for File Services 93
Transient Data Services. 97
Dispose of Data (PUT). 99
Acquire Queued Data (GET). 101
Control the Processing of an Extrapartition Data Set (FEOV). . 104
Purge Intrapartition Data (PURGE). 105
Test Response to a Request for Transient Data Services 105
Temporary Storage Services 107
Store or Update Temporary Data (PUT or PUTQ) 109
Retrieve Temporary Data (GET or GETQ). 111
Free Temporary Data (RELEASE or PURGE) 112
Test Response to a Request for Temporary Storage Services. . . 113
Storage Services 116
Obtain and Initialize Main Storage (GETMAIN) 116
Release Main Storage (FREEMAIN). 118
Program Services 119
Pass Program Control Anticipating Return (LINK). 120
Transfer Program Control (XCTL). 121
Load a Program (LOAD). 122
Return Program Control (RETURN). 123
Delete a Loaded Program (DELETE) 124
Abnormally Terminate a Transaction (ABEND) 125
Activate, Cancel, or Reactivate an Exit for Abnormal
Termination Processing (SETXIT or RESETXIT) 126
Convert Symbolic Label to Address (COBADDR). 127
Test Response to a Request for Program Services. 127
Time Services. 130

Time-of-Day Services (GETIME) 131
Time-Ordered Task Synchronization (WAIT or POST) 132
Delay the Processing of a Task (WAIT) 132
Signal the Expiration of a Specified Time (POST) 134
Automatic Time-Ordered Task Initiation (INITIATE or PUT) . . . 136
Task Initiation without Data (INITIATE) 136
Task Initiation with Data (PUT) 137
Retrieve Time-Ordered Data (GET) 139
Time-Ordered Request Cancellation (CANCEL) 141
Cancel an Interval Control POST Request. 141
Cancel an Interval Control WAIT Request. 142
Cancel an Interval Control INITIATE or PUT Request 142
Input/Output Error Retry Capability (RETRY) 142
Test Response to a Request for Time Services 142
Task Services. 144
Initiate a Task (ATTACH) 145
Reschedule a 3650 Task (SCHEDULE) 148
Change Priority of a Task (CHAP) 149
Synchronize a Task (WAIT) 150
Synchronize a Task with a Single Event 150
Synchronize a Task with One of a List of Events. 151
Relinquish Control to a Task of Higher Priority. 152
Single-Server Resource Synchronization (ENQ/DEQ) 152
Declare the Purgeability of a Task on System Overload (PURGE/NOPURGE) 155
Journal Services 156
Acquire the Journal Control Area (GETJCA) 157
Create a Journal Record and Wait for Output (PUT) 159
Create a Journal Record for Asynchronous Output (WRITE) 161
Synchronize with the Output of a Journal Record (WAIT) 165
Test Response to a Request for Journal Services. 167
Recovery/Restart Services. 169

CHAPTER 7. SYSTEM MANAGEMENT MACRO INSTRUCTIONS - GENERAL FORMATS. 171
DFHTC Macro Instruction. 171
Input Operations 171
Output Operations. 177
Miscellaneous Operations 186
DFHFC Macro Instruction. 188
Randomly Retrieve Data from a Data Set 188
Randomly Update or Add Data to a Data Set. 191
Randomly Delete Data from a Data Set (VSAM Only) 192
Obtain a File Work Area. 194
Release File Storage 195
Initiate Sequential Retrieval (Browsing) 196
Retrieve Next Sequential Record. 199
Terminate Sequential Retrieval (Browsing) 200
Reset Sequential Retrieval 200
Test Response to a Request for File Services 202
DFHTD Macro Instruction. 206
Dispose of Data. 207
Acquire Queued Data. 207
Control Processing of Extrapartition Data Set (Magnetic Tape Only) 208
Purge Intrapartition Data. 208
Test Response to a Request for Transient Data Services 209
DFHTS Macro Instruction. 210
Store Temporary Data as a Single Unit of Information 210
Store Data to a Temporary Storage Message Set. 211
Retrieve a Single Unit of Temporary Data 213
Retrieve Data from a Temporary Storage Message Set 214
Release a Single Unit of Temporary Data. 216
Purge a Temporary Storage Message Set. 216
Test Response to a Request for Temporary Storage Services. . . 217

DFHSC Macro Instruction. 218
Obtain and Initialize Main Storage 218
Release Main Storage 220
DFHPC Macro Instruction. 220
Pass Program Control Anticipating Return 220
Transfer Program Control 221
Load a Program 222
Return Program Control 222
Delete a Loaded Program. 223
Abnormally Terminate a Transaction 223
Activate or Cancel an Exit for Abnormal Termination Processing. 224
Reactivate an Exit for Abnormal Termination Processing 225
Convert Symbolic Label to Address. 225
Test Response to a Request for Program Services. 226
DFHIC Macro Instruction. 226
Time-of-Day Updating 226
Delay Processing of a Task 228
Signal Expiration of a Specified Time. 230
Initiate a Task. 231
Task Initiation with Data. 233
Retrieve Time-Ordered Data 235
Cancel a Request for Time Services 236
I/O Error Retry. 237
Test Response to a Request for Time Services 238
DFHKC Macro Instruction. 239
Initiate a Task. 239
Reschedule a 3650 Task 240
Change Priority of a Task. 240
Synchronize a Task 240
Enqueue upon a Resource. 241
Dequeue upon a Resource. 242
Declare a Task to be Purgeable 242
Declare a Task to be Nonpurgeable. 243
DFHJC Macro Instruction. 243
Acquire the Journal Control Area (JCA) 243
Create a Journal Record and Wait for Output. 244
Create a Journal Record. 246
Wait for Output of a Journal Record. 248
Test Response to a Request for Journal Services. 249
DFHSP Macro Instruction. 250

CHAPTER 8. PROGRAM TESTING AND DEBUGGING. 251
Sequential Terminal Support. 251
Trace Services 253
Trace ON Function. 254
Trace OFF Function 255
Trace ENTRY Function 257
Trace Table. 259
Dump Services. 287
Dump Transaction Storage (TRANSACTION) 288
Dump CICS/VS Storage (CICS). 289
Dump Transaction Storage and CICS/VS Storage (COMPLETE). . . . 289
Dump Partial Storage (PARTIAL) 290

CHAPTER 9. CICS/VS BUILT-IN FUNCTIONS 295
Table Search 295
Phonetic Conversion. 295
Verification of a Data Field 296
Editing of a Data Field. 296
Bit Manipulation 296
Input Formatting 296
Fixed Format 296
Positional Format. 297
Keyword Format 297

Combination Input. 298
Weighted Retrieval 298
DFHBIF Macro Instructions. 302
DFHBFTCA Macro Instruction 302
Table Search 303
Returned Values. 306
Example - Separate Tables. 306
Example - Complex Table. 307
Phonetic Conversion. 308
Returned Value 308
Phonetic Coding Method 309
Examples 309
Phonetic Conversion Subroutine 309
Field Verify 310
Returned Values. 311
Example. 312
Field Edit 312
Returned Value 313
Example. 313
Bit Manipulation 313
Returned Values. 315
Examples 315
Input Formatting 315
Storage Definition 316
DFHBIF TYPE=DEFLDNM Macro Instruction. 316
Required Delimiters. 317
DFHBIF TYPE=INFORMAT Macro Instruction 317
Returned Values. 318
Examples 318
Weighted Retrieval 320
Initiate Weighted Retrieval. 320
Returned Values. 323
Establish Selection Criteria 323
Retrieve Selected Records. 326
Returned Values. 327
Release Weighted Retrieval Storage Areas 327
Test Response to a Request for Weighted Retrieval. 328
Example. 330

CHAPTER 10. BASIC MAPPING SUPPORT 333
BMS Advantages 333
Device Independence. 333
Format Independence. 333
BMS Techniques 334
Data Mapping and Formatting. 334
Terminal Paging. 335
Message Routing. 336
Programming Considerations 337
Specifying Maps for Basic Mapping Support. 338
Implied READ/WRITE 338
Map Definition 339
Input Mapping. 339
Output Mapping 340
Input/Output Mapping 341
Offline Map Building 341
DFHMSD Macro Instruction 341
DFHMDI Macro Instruction 346
DFHMDF Macro Instruction 350
Online Map Use 360
Establishing Addressability to User-Supplied Data. 360
DFHBMS Macro Instruction 361
Input Operations 361
Output Operations. 363
Cumulative Page Building with Mapping. 364
PAGEBLD Overflow Processing. 370

Cumulative Page Building without Mapping 373
Non-Cumulative Page Building 378
Terminating a Logical Message. 384
BMS Message Recovery 388
Abnormally Terminating a Logical Message 388
Message Routing. 388
Disposition and Message Routing. 393
Test Response to a Request for BMS Services. 394
Terminal Code (TC) Table 399
Status Flag Byte in User-Supplied Route List 400
Standard Attribute List and Printer Control Characters (DFHBMSCA). 401
Standard Attention Identifier List (DFHAID). 402
BMS TIOA Specification 403
Programming Considerations for Paging Commands on Video Devices 405
Examples of the Use of BMS 405

CHAPTER 11. APPLICATION PROGRAMMING CONSIDERATIONS. 415
Terminal-Oriented Task Identification. 415
Programmable Device Considerations 416
3735 Considerations. 417
3735 Transactions - Autoanswer 418
3735 Transactions - Autocall 418
3740 Considerations. 418
Batch Mode Applications. 418
Inquiry Mode Applications. 419
System/7 Considerations. 419
Non-Programmable Device Considerations 421
2260/2265 Programming Considerations 421
3270 Operating in 2260 Compatibility Mode. 422
2770/2780 Programming Considerations 423
2980 Programming Considerations. 423
Passbook Control 423
Segmented Writes Control 424
Data Handling. 425
Writing High-Level-Language Programs 425
7770 Programming Considerations. 429
2741 Read Attention and Write Break Support. 430
Read Attention (CICS/OS/VS or CICS/DOS/VS) 430
Write Break (CICS/OS/VS) 431
3270 Print Function. 431
Teletypewriter Programming Considerations. 432
Message Format 432
Message Length 432
Asynchronous Transaction Processing. 432
Data Base Considerations 432
Segmented Records. 433
Segmented Record Formats 434
Segment Indicators 435
Main Storage Processing of Segmented Records 437
Segment Sets 438
Indirect Accessing 439
Duplicate Records. 441
Record Identification Field. 444
Updating Nonkeyed DAM Data Sets. 446
Adding Records to DAM Data Sets. 446

CHAPTER 12. REQUESTING DATA LANGUAGE/I SERVICES 449
Quasi-Reentrant Considerations with Regard to DL/I 449
Obtaining Addresses of Program Communication Blocks. 450
DFHFC Macro Instruction (CICS/OS/VS) 450
DL/I CALL Statement (CICS/DOS/VS). 451
Building Segment Search Arguments. 452
Acquiring an I/O Work Area 453

Issuing the DL/I CALL. 454
DFHFC Macro Instruction (CICS/OS/VS) 454
DL/I CALL Statement (CICS/OS/VS or CICS/DOS/VS). 456
Releasing a PSB in the CICS/VS Application Program 457
DFHFC Macro Instruction (CICS/OS/VS) 457
DL/I CALL Statement (CICS/DOS/VS). 458
Checking the Response to a Request for DL/I Services 459
DL/I Requests in an Assembler-Language Program (CICS/OS/VS). . 462
DL/I Requests in an ANS COBOL Program (CICS/OS/VS) 464
DL/I Requests in a PL/I Program (CICS/OS/VS) 466

APPENDIX A. EXECUTABLE CICS/VS SAMPLE PROGRAMS. 469

APPENDIX B. SUMMARY OF CICS/VS STORAGE AREAS. 481

APPENDIX C. EXPLANATIONS OF MNEMONICS 485

APPENDIX D. CICS/VS MACRO INSTRUCTIONS. 497
Terminal Services. 497
File Services. 482
Transient Data Services. 485
Temporary Storage Services 486
Storage Services 488
Program Services 489
Time Services. 490
Task Services. 492
Journal Services 493
Restart/Recovery Services. 494
Trace Services 495
Dump Services. 496
Built-In Functions 497
Basic Mapping Support Services 518
CICS/VS-DL/I Interface Services. 522

APPENDIX E. TRANSLATE TABLES FOR THE 2980 523

BIBLIOGRAPHY 533

INDEX. 535

FIGURES

1-1. Conventional Batch Processing 1
1-2. Transaction Processing of CICS/VS 2
1-3. CICS/VS Data Base Concept 3
1-4. CICS/VS Transaction Flow. 6
1-5. A Comparison of Batch and Online Environments 8
1-6. Register Usage Under CICS/VS. 13
2-1. CICS/VS Storage Areas 19
2-2. Symbolic Names and Base Addresses of CICS/VS Storage Areas 22
2-3. Chaining of CICS/VS Storage Areas 23
3-1. Example of CICS/VS Assembler-language Application Program 38
4-1. Example of CICS/VS ANS COBOL Application Program. 48
5-1. Example of CICS/VS PL/I Application Program 57
6-1. File Control Response Codes 95
6-2. Use of DFHTD TYPE=PUT 100
6-3. Use of DFHTD TYPE=GET 102
6-4. Transient Data Control Response Codes 104
6-5. Temporary Storage Control Response Codes. 114
6-6. Communication and Logical Relationships Among Application Programs. 120
6-7. Program Control Response Codes. 128
6-8. Interval Control Response Codes 143
6-9. Task Synchronization Under CICS/VS. 148
6-10. Journal Control Response Codes. 168
8-1. Trace Table Entry for Task Control. 261
8-2. Trace Table Entry for Storage Control 262
8-3. Trace Table Entry for Program Control 264
8-4. Trace Table Entry for Interval Control. 265
8-5. Trace Table Entry for Dump Control. 267
8-6. Trace Table Entry for File Control. 268
8-7. Trace Table Entry for Transient Data Control. 270
8-8. Trace Table Entry for Temporary Storage Control 271
8-9. Trace Table Entry for CICS/VS-DL/I Interface. 272
8-10. Trace Table Entry for Journal Control 273
8-11. Trace Table Entry for Basic Mapping Support 274
8-12. Trace Table Entry for Built-In Functions. 276
8-13. Trace Table Entry for VTAM Terminal Control 278
8-14. Trace Table Entry for Trace Control 285
8-15. Trace Table Entry for Sync Point Program. 2865
8-16. Trace Table Entry for Field Engineering (FE) Type of Entry. 286
8-17. Trace Table Entry for Task Control (During Auxiliary Trace Only) 286
9-1. Selection of records by Weighted Retrieval. 300
9-2. Table Search Response Codes 307
9-3. Field Verify Function Response Codes. 312
9-4. Bit Manipulation Response Codes 315
9-5. INFORMAT Response Codes 319
9-6. Weighted Retrieval Response Codes 330
10-1. Use of Trailer Maps in PAGEBLD Mapping Operations 371
10-2. Overflow Processing by Application Programs Under BMS . . 372
10-3. BMS Response Codes. 398
10-4. BMS Terminal Code Table 399
10-5. BMS Status Flags. 400
10-6. 3270 Field Attributes and Printer Control Characters. . . 402
10-7. 3270 Attention Identifiers and Functions. 402
10-8. Symbolic Storage Definition Input 406
10-9. Symbolic Storage Definition Using LANG=ASM,MODE=INOUT . . 407

10-10. Symbolic Storage Definition Using LANG=ASM,MODE=IN. . . . 408
10-11. Symbolic Storage Definition Using LANG=ASM,MODE=OUT . . . 409
10-12. Symbolic Storage Definition Using LANG=COBOL,MODE=INOUT . 410
10-13. Symbolic Storage Definition Using LANG=COBOL,MODE=IN. . . 411
10-14. Symbolic Storage Definition Using LANG=COBOL,MODE=OUT . . 412
10-15. Symbolic Storage Definition Using LANG=PLI,MODE=INOUT . . 413
10-16. Symbolic Storage Definition Using LANG=PLI,MODE=IN. . . . 414
10-17. Symbolic Storage Definition Using LANG=PLI,MODE=OUT . . . 414
11-1. CICS/VS terminal-oriented task identification 416
12-1. CICS/VS-DL/I Interface response codes 460

SUMMARY OF AMENDMENTS

This Technical Newsletter (SN20-9086) discusses the following enhancements for the CICS/VS application programmer:

- Data recovery for Interval Control

  The application programmer now has the option of keeping or releasing Interval Control records for temporary storage.

- DL/I recovery

  A sync point request for a DL/I resource implies the release of that resource.

- IBM 3275 Display Station, dial

  CICS/VS support is extended to 3275 dial.

- IBM 3650 Retail Store System

  The application programmer can use CICS/VS functions, including BMS, to converse with the 3650.

- Teletypewriter Support (World Trade only)

  CICS/VS support is extended to the Teletypewriter (for World Trade countries only).

- BMS message recovery

  CICS/VS provides message recovery for logical BMS message that are stored on temporary storage.

- Auxiliary Trace

  CICS/VS enables the user to write trace records to an auxiliary data set.

Significant changes in the text of the manual are marked with a vertical bar in the margin to the left of the changed section.

Chapter 6	Chapter 7
TERMINAL SERVICES	DFHTC
FILE SERVICES	DFHFC
TRANSIENT DATA SERVICES	DFHTD
TEMPORARY STORAGE SERVICES	DFHTS
STORAGE SERVICES	DFHSC
PROGRAM SERVICES	DFHPC
TIME SERVICES	DFHIC
TASK SERVICES	DFHKC
JOURNAL SERVICES	DFHJC
RESTART/RECOVERY SERVICES	DFHSP

CHAPTER 1. INTRODUCTION

The IBM Customer Information Control System/Virtual Storage (CICS/VS) is a transaction-oriented data base/data communication system. It can be applied to most online IBM System/370 systems, since it offers terminal facilities for many applications: message switching, inquiry, data collection, order entry, and conversational and batched data entry.

CICS/VS works with either the Disk Operating System/Virtual Storage (DOS/VS) or the Operating System/Virtual Storage (OS/VS1 or OS/VS2). It can be thought of as an extension of the operating system or as an interface between the operating system and the user's application programs. The system is modular: at system generation or initialization, an installation can select the components it needs to tailor a CICS/VS system for a given application.

In conventional batch processing (see Figure 1-1), like transactions are grouped for processing, and the application programmer plans a series of runs to edit input transactions, update data sets, or write output reports. Because the programmer concentrates on carefully manipulating data for most efficient handling of each transaction type, the data in batch processing becomes closely tied to the program logic and has little value for other applications.

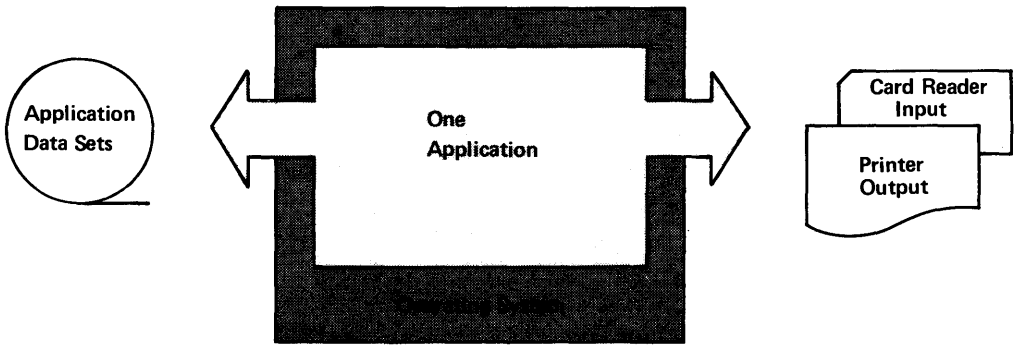


Figure 1-1. Conventional Batch Processing

Real-time data base/data communication (DB/DC) differs from batch processing primarily in the number and types of activities taking place in the system concurrently. A batch-processing system schedules each application independently and provides data base support unique to each application. A DB/DC system controls many random nonscheduled transactions for many applications and provides one integrated data base supporting all the applications on the system (see Figure 1-2).

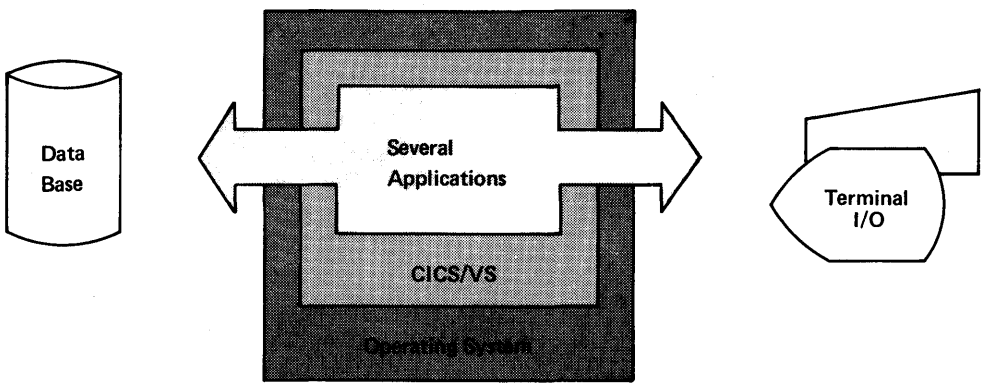


Figure 1-2. Transaction Processing of CICS/VS

The CICS/VS program product (either CICS/OS/VS or CICS/DOS/VS) performs numerous functions essential to success in real-time data base/data communications. Its major responsibilities can be summarized as follows:

- Provides rapid response to simultaneously active online terminals
- Controls a telecommunication network of mixed devices
- Concurrently manages a wide mixture of transactions being serviced by a variety of programs
- Controls access to a data base
- Effectively manages system resources, such as dynamic storage, to keep the system in continuous operation
- Assigns priorities to optimize the use of the CPU processing facility

With these system functions assumed by CICS/VS, application programmers can concentrate on their particular applications. Programming takes less time, debugging is easier, and implementation time and costs are reduced accordingly.

A key consideration in selecting a data base/data communication system is its adaptability to present and future needs. CICS/VS is a family of systems that provides a DB/DC interface to IBM System/370 at most levels of the product line, offering a clear path for growth or migration of an installation.

Figure 1-3 shows how the CICS/VS data base supports the information needs of multiple applications, independently and concurrently.

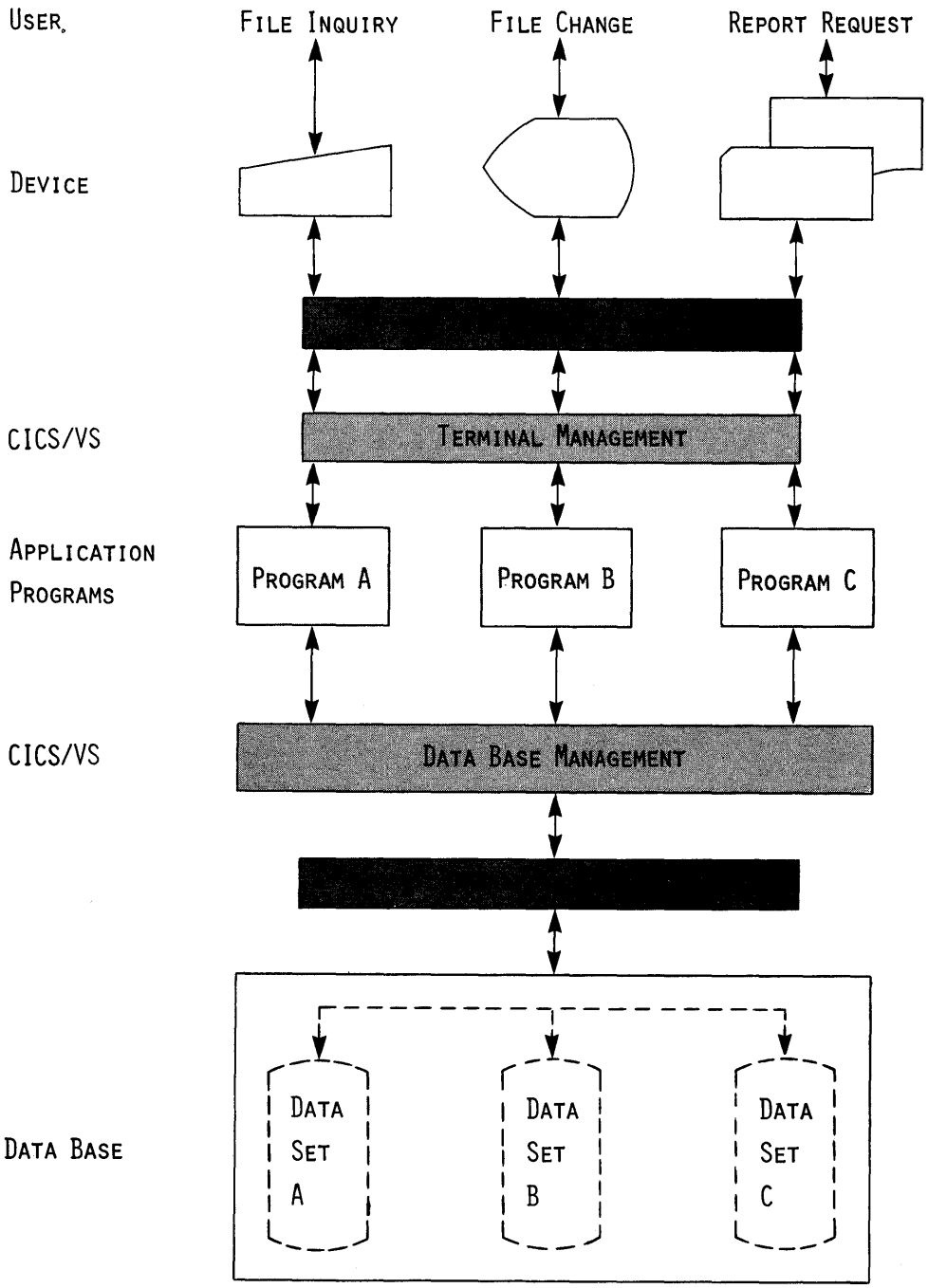


Figure 1-3. CICS/VS Data Base Concept

Although application programmers need not be concerned with details of CICS/VS structure or performance, they should have a general understanding of how CICS/VS components interact to perform essential processing steps. CICS/VS consists of six major components, explained in greater detail in the Customer Information Control System (CICS/VS) General Information Manual.

- System management
- System services
- System monitoring
- System reliability
- System support
- Application services

Each of these components is divided into functions which provide services to CICS/VS users. The components that most directly affect the application programmer are system management, system monitoring, and system reliability. To help the application programmer understand some of the ways in which CICS/VS assists him, the system management functions are summarized below (see Chapter 6 for further details).

- Terminal management - provides for communication between terminals and user-written application programs through the terminal central program. This facility supports automatic task initiation to process new transactions. The testing of application programs is accommodated by the simulation of terminals through sequential devices such as card readers, line printers, or tape units, or disk storage units.

- File management - provides a data base facility using direct access, indexed sequential, and virtual storage data management. This function supports updates, additions, random retrieval, and selective retrieval (browsing) of logical data on BDAM, ISAM, and VSAM data base data sets. Additional capabilities provided for only VSAM data sets include record deletion, skip-sequential processing, key-ordered mass insertion, relative byte addressing, search key high/equal, generic key, and locate mode processing for read-only requests. Optional access to the Data Language/I (DL/I) facility of the IBM Information Management System/Virtual Storage (IMS/VS) is provided under CICS/OS/VS. Such use of DL/I requires installation of the IBM program product IMS/VS Data Base System (5740-XX2).

  Note: Users of CICS/DOS/VS can interface with the IBM program product DOS/VS DL/I (5746-XX1) through DOS/VS DL/I CALLs, but CICS/VS file control macro instructions cannot be utilized.

- Transient data management - provides the optional queuing facility for the management of data in transit to and from user-defined destinations. This function facilitates message switching and data collection.

- Temporary storage management - provides the optional general-purpose "scratch pad" facility. This facility is intended for video display paging, broadcasting, data collection suspension, conservation of dynamic storage, retention of control information, and similar functions. Where multiple records are used and random access to those records is necessary, this function also provides a queuing facility.

- Storage management - controls dynamic storage allocated to CICS/VS. Storage acquisition, disposition, initialization, and request queuing are among the services and functions performed by this component of CICS/VS.

- Program management - provides a multiprogramming capability through

dynamic program management while offering a real-time program fetch capability.

- Time management - provides control of various task functions (for example, runaway task control, task synchronization, and system stall detection) based on specified intervals of time or the time of day.

- Task management - provides the dynamic multitasking facilities necessary for effective, concurrent transaction processing. Functions associated with this facility include priority scheduling, transaction synchronization, and control of serially reusable resources. This function controls activities within the CICS/VS partition or region and is in addition to the multitasking or multiprocessing capabilities of the host operating system.

- Journal management - provides facilities for creating and managing special-purpose sequential data sets, called journals, during real-time execution of CICS/VS. Journals are intended for recording (in chronological order) any data that the user may need in subsequent reconstruction of data or events. Examples of such data sets are an audit trail, a change-file of data base updates and additions, and a record of system transaction-activity (often called a log).

- Sync point management - works in conjunction with other CICS/VS components, such as transient data management and file management, to provide the facilities needed for an emergency restart of CICS/VS after abnormal termination. The CICS/VS transaction backout program (DFHTBP) or user programming can accomplish changes to data base data sets or transient data intrapartition queues for tasks "in-flight" at time of failure on the basis of system information recorded on a system log during online execution of CICS/VS.

In addition to these supervisory and data management functions, CICS/VS provides dump management and trace management, which are especially valuable to the application programmer in program debugging. CICS/VS basic mapping support (BMS) facilitates information display on a wide variety of terminals and provides device independence, terminal paging, and message routing capabilities. Numerous built-in functions are available for use by application programs. CICS/VS also provides system service programming to identify terminal operators, to give dynamic control of the entire system to a master terminal, to display real-time system statistics, to intercept abnormal conditions not handled directly by the operating system, and to end operation by gathering summary statistics, closing data sets, and returning control to the operating system.

To achieve its objective of providing rapid response to terminal users, CICS/VS executes in a multitasking mode of operation within its own partition or region. Such multitasking within one partition or region is analogous to multiprogramming within the total DOS/VS or OS/VS environment. Generally, tasks are initiated as a result of transactions entered at terminals. Whenever one task is forced to wait for completion of an I/O operation, availability of a resource, or some other cause of delay, processing of another task within the system is initiated or continued.

The interrelationships of, and services performed by, various CICS/VS System Management functions in the processing of a transaction (task) are shown in Figure 1-4. Some general characteristics of application programs to be run under CICS/VS and the use of other functions that it provides are explained in subsequent sections of this manual.

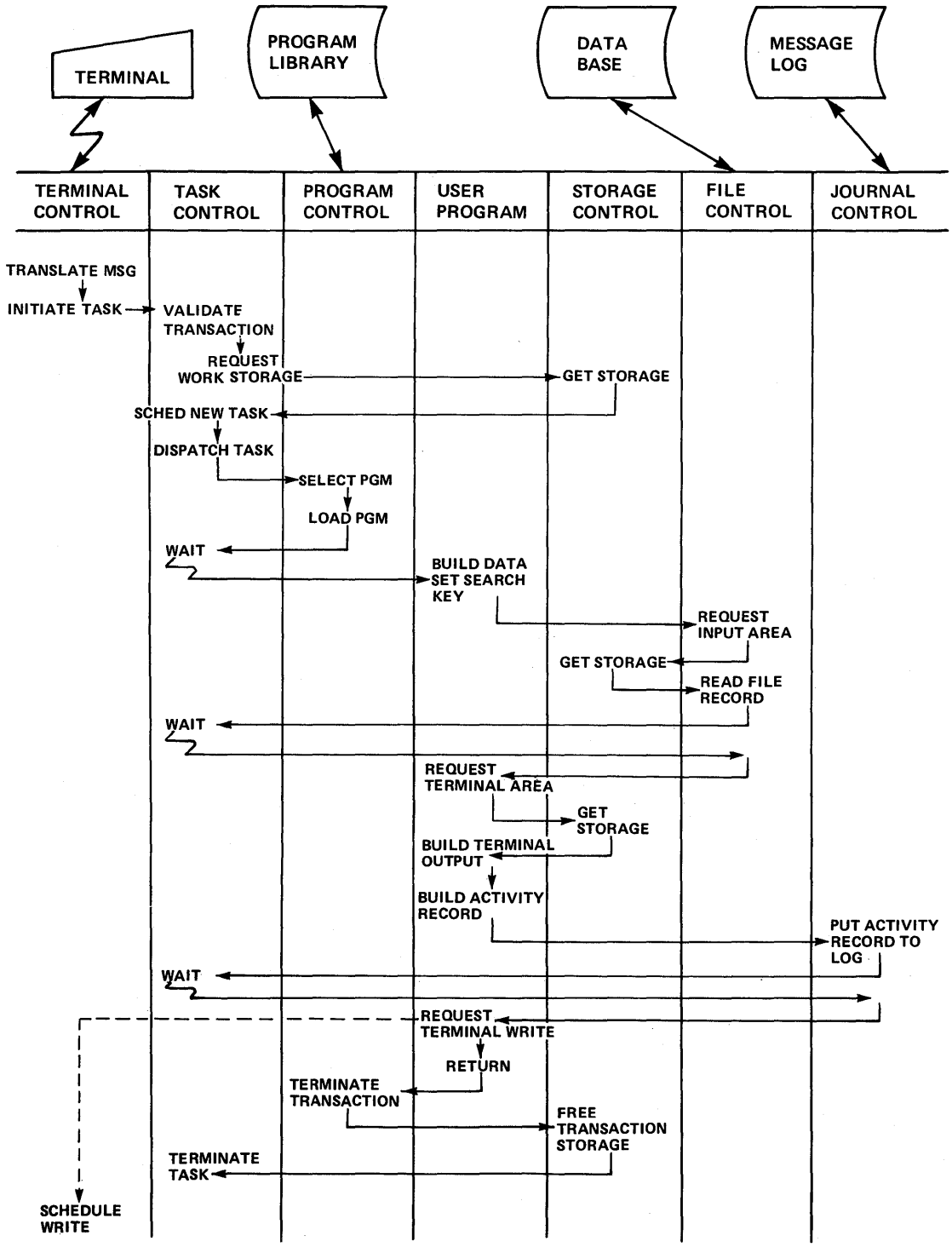


Figure 1-4. CICS/VS Transaction Flow

PROGRAMMING TECHNIQUES

Programs to be run under CICS/VS may be coded in Assembler language, American National Standard (ANS) COBOL, or PL/I. Writing a program to be run under CICS/VS is not significantly different from writing a program to be run on any of numerous computing systems. However, the

CICS/VS user should be aware that CICS/VS is an online system and that programs running under CICS/VS operate in an online environment. Some of the basic differences between online systems and the traditional batch-processing environment are summarized in Figure 1-5.

Single threading is the execution of a program to process inputs to completion, sequentially. Processing of one input is completed before another input is acted upon. In contrast, multithreading is the capability of using various sections of a single program concurrently. Under CICS/VS, for example, the first section of an application program may be executing to process one transaction. When that section is completed (in general, signaled by executing a CICS/VS macro instruction that causes a wait), processing of another transaction using a different section of code in the same program may ensue. Just as there is not usually one clearly superior, correct way to solve a problem, so there is not usually one correct way to write a program to implement that solution. Nevertheless, there are good and bad techniques of programming under CICS/VS. How much time and thought should be given to programming style when writing a program? The answer depends largely on the expected usage of the program. Will it be used once a day or once a year? When used, will it run for two minutes or five hours? The frequency and length of use are important factors to consider when deciding how much time to spend on programming techniques (that is, to devising the optimum solution to a problem).

Some of the basic characteristics of application programs to be run under CICS/VS are summarized below. These characteristics should be viewed as essential to successful operation under CICS/VS (although some are not mandatory, they are highly advisable).

1. Programs must be quasi-reentrant (see "Quasi-reentrance" in this chapter).

2. CICS/VS macro instructions (rather than programming-language statements such as READ, GET, PUT, or WRITE) are included to control numerous functions required in application programs. (See "CICS/VS Macro Instructions" in this chapter.)

3. Input/output areas, temporary storage areas, and work areas are not included in an application program. All or portions of these areas are defined outside of application programs. The application programmer must work with CICS/VS system programmers in defining these areas by means of tables within CICS/VS. (See "Storage Definition" in this chapter and Chapter 2.)

4. Files are not defined within application programs. As in item 4, the application programmer works with CICS/VS system programmers in establishing these definitions. (See the CICS/VS System Programmer's Reference Manual and applicable OS/VS or DOS/VS publications.)

5. The application programmer must establish addressability in his program to CICS/VS storage areas accessed by his program.

6. Working storage should not be tied up, for example, awaiting a reply from a terminal user.

7. Programs should be as efficient as possible, without overemphasizing this goal, to work with CICS/VS in providing rapid response to terminals.

Batch Processing	Characteristic	Online Application
Generally sequential from cards, tape, or a direct access storage device (DASD); submitted as groups of related data, edited, and verified	Input	Random, multiple, concurrent but unrelated entries from terminals; immediate edit and verification of each entry
Sequential, generally single-thread, with updating of sequential master files	Processing	Random, multithreading, as one aspect of multitasking within a partition or region; for inquiry or updating purposes or both
Generally in the form of updated master files and printed reports	Output	Messages to terminals updated files, and system log of activities
Start Read transaction ←┐ Read master Process ──┘	Sequence of operations	System initialized then transactions processed as occur, with data rather than program as driver
Signaled by last transaction	End of job	Generally, end of shift or day
Predictable, known before run	Amount of activity	Not predictable, tend to fluctuate widely
Applications "own" master files on tape or DASD; placed online when required for run	Master files/ data sets	Files accessible to multiple, authorized applications; must be online; are on DASD
Varies widely; usually involves manual procedures	Response time	Measured in seconds; generally occurs as message to terminal

Figure 1-5. A Comparison of Batch and Online Environments

The general structure of a CICS/VS application program can be summarized as follows:

- Storage definition statements - see "Storage Definition" in this chapter and Chapters 2 through 5
- Program initialization statements - see "Program Initialization" in this chapter and requirements for establishing addressability in Chapters 2 through 5
- Processing statements - see "CICS/VS Macro Instructions" in this chapter and Chapters 6 and 7; refer to Chapters 8, 9, 10, and 12 if additional functions are desired
- Termination statements - see "CICS/VS Macro Instructions" in this chapter and Chapters 6 and 7

No attempt is made in this text to teach the use of typical programming-language statements or general programming techniques within Assembler language, ANS COBOL, or PL/I. Documentation for these languages should be consulted for such information (see the Bibliography).

APPLICATION PROGRAM PACKAGING

Application design for a virtual environment is both similar and different from application design in a real environment. The system should have all modules resident so that code on un-referenced page(s) need not be paged in. If the program is dynamic, the entire program must be loaded across adjacent pages before execution begins. Dynamic programs can be purged from storage if not currently in use and an unsatisfied storage request exists. To allow sufficient dynamic area to prevent the purging is more expensive than making them resident since a dynamic program will not share unused space on a page with another program.

The reference pattern of the application should touch the fewest concurrent pages during its execution.

1. The main line execution should be as straight a line as possible. The ideal program executes sequentially with no branch logic referencing beyond a small range of address space.
2. Literals and subroutines should be coded as close to their use as possible. This would include LTORG statements at appropriate locations in the program. Place single used constants near its code. Executed instructions should be near the EX instruction. Perform and GO TO routines should be placed near the caller.
3. Avoid use of COBOL EXAMINE or VARIABLE MOVE operations since these expand into subroutine executions.
4. Do not alter anything within the program module. An unchanged module is reentrant and is not paged out.
5. Use the TWA for changeable data during execution, i.e., counters, switches, parameter passing, basic mapping support output area (use BMS SAVE).
6. Do few or no user GETMAINS to minimize the task's reference pattern.

7. Do not modularize for the sake of size as was recommended for pre-VS systems. Consider duplicate code in line as opposed to subroutines or separate modules.

8. Avoid LINKs since it will cause a GETMAIN for a RSA and will search the PPT.

9. Try to keep the execution path straight line by using XCTL.

QUASI-REENTRANCE

Application programs must be coded so that they are "serially reusable" between entry and exit points of the program. A serially reusable portion of an application program is executed by only one transaction at a time, and must initialize and/or restore any instructions or data that it alters within itself during execution. (It is recommended, however, that all applications be truly reentrant to minimize paging.) Entry and exit points coincide with the use of CICS/VS macro instructions, since an application program loses control only upon execution of a CICS/VS macro instruction.

This required quality of application programs written to run under CICS/VS is called "quasi-reentrance," since the programs need not meet System/370 specifications for true reentrance. Quasi-reentrance allows a single copy of a user-written application program to be used to process several transactions concurrently, thereby reducing the number of copies of a program that must be in main storage.

Intermediate exits may be taken during execution of an application program. Such exits constitute a transfer of control from the program. All switches, data, and intermediate results needed upon subsequent return to the program must be retained in a unique storage area such as the transaction work area (TWA). The application programmer must provide that unique intermediate storage area by symbolically defining it in his program (as described under "Symbolic Storage Definitions" in Chapter 2).

A serially reusable application program that has no intermediate exits also has the quality of quasi-reentrance.

CICS/VS MACRO INSTRUCTIONS

As stated earlier, application programs to be executed under CICS/VS can be written in Assembler language, American National Standard (ANS) COBOL, or PL/I. Regardless of the language used, it is strongly recommended that the application programmer allow CICS/VS to perform all supervisory and data management services for his applications by using CICS/VS macro instructions to invoke the desired services. He should use similar macro instructions to request dump and trace facilities when testing or debugging his programs. Although the application programmer is not precluded from direct communication with the operating system, the results of such action are unpredictable and performance may be affected. Such action also has a limiting effect on migration from CICS/DOS/VS to CICS/OS/VS, a growth path that may become highly advisable for the CICS/DOS/VS user.

CICS/VS macro instructions are written in Assembler language in the following format:

Name	Operation	Operands	Comments
blank or symbol	DFHxxxxx	One or more operands separated by commas	Comments for program documentation

The name field of a CICS/VS macro instruction must not contain a label if the macro instruction is used in an ANS COBOL or PL/I program; if a label is desired for the macro instruction, it may be placed on the line preceding the macro instruction. For COBOL programs, the first six positions may contain a sequence number.

The operation field of a CICS/VS macro instruction must begin before column 16 and must contain the three-character combination "DFH" in the first three positions of the operation field. Up to five additional characters can be appended to DFH to complete this symbolic name for the appropriate program or table. Since DFH is reserved for CICS/VS macro instructions, no other statement may begin with this three-character combination.

The operand field of a CICS/VS macro instruction is used to specify the services and options to be performed. The following general rules apply:

1. Operands that are written in uppercase letters (for example, TYPE=INITIAL) are to be coded exactly as shown.

2. Operands that are written as a combination of uppercase and lowercase letters separated by an equal sign are to be coded with the keyword on the left as it appears and an appropriate substitution for the general class of elements on the right. For example, if the format description contains NORESP=symbolic address, the user may code NORESP=NORMROUT.

3. Commas and parentheses are coded as shown. However, the parentheses are required only when multiple operand entries are used. For example, the following coding is correct:

   ```
   TYPE=READ
   TYPE=(READ,WAIT)
   ```

   The commas are used as separators, but no comma should precede the first operand entry or follow the last one inside parentheses. Similarly, no comma should follow the last operand coded for a particular macro instruction.

4. Since a blank character indicates the end of the operand field, the operand field must not contain blanks except after a comma on a continued line or after the last operand of the macro instruction. The first operand on a continuation line must begin in column 16.

5. When a CICS/VS macro instruction is coded on more than one line, each line containing part of the macro instruction (except the last one) must contain a nonblank character (for example, an asterisk) in column 72 indicating that the macro instruction has been continued on the next line.

CODING AIDS

Throughout this manual, wherever a CICS/VS macro instruction is presented, the symbols { }, |, [ ], and ... are used in defining the instruction format. These symbols are not part of the macro instruction and are not coded by the programmer. Their purpose is to indicate how the macro instruction may be written, they should be interpreted as follows:

1. Braces { } are used to denote grouping. An example is:

   ```
   SEGSET={symbolic address}
          {YES             }
          {ALL             }
   ```

   The vertical stacking indicates that a choice is to be made. The above example indicates that the coding SEGSET= must be followed by a programmer-selected symbolic address, the keyword YES, or the keyword ALL.

2. The vertical stroke | indicates that a choice is to be made. It is the same as the use of the word "or." For example,

   ```
   [,INTRVAL={numeric value}] | [,TIME={numeric value}]
             {YES           }         {YES           }
   ```

   means that either the INTRVAL operand or the TIME operand, but not both, can be specified in the macro instruction. That is, INTRVAL and TIME are mutually exclusive operands.

3. Square brackets [ ] denote options. Anything enclosed in brackets may or may not be coded, depending on whether or not the associated option is desired. An example is:

   ```
   [MODE={MOVE  }]
         {LOCATE}
   ```

   If a default value is assumed by CICS/VS in the case of an omitted operand, that default value is indicated by underlining (see MOVE above).

4. An ellipsis (three dots ...) denotes that the immediately preceding unit may appear one or more times in succession in the macro instruction.

Chapter 6 explains how CICS/VS macro instructions are used to request supervisory and data management services. All CICS/VS macros the application programmer may use are fully defined in Chapter 7 and summarized in the listing of Appendix D.

STORAGE DEFINITION

The macro library supplied with CICS/VS contains symbolic storage definitions of CICS/VS control areas, work areas, and I/O areas. It is strongly recommended that the application programmer use these definitions rather than develop actual or direct displacements in his program. This protects the application program in the event of any relocation of CICS/VS.

The Assembler-language programmer includes symbolic storage definitions in his program by means of Assembler-language COPY statements. For the PL/I programmer, the macro library contains numerous BASED structures that describe CICS/VS control areas. These dummy sections are available to the user through use of %INCLUDE

statements. The ANS COBOL programmer uses similar definitions through COPY statements in the Linkage Section of the Data Division of his application program. These definitions are discussed in Chapter 2.

PROGRAM INITIALIZATION

In the initialization section of the application program, the Assembler-language programmer must establish a symbolic base address for his program because this is not done by CICS/VS prior to entry. In doing so, he identifies a base register. Register 12 is reserved by CICS/VS for the address of the task control area (TCA) for this task. Register 13 is reserved for the address of the common system area (CSA). These registers are initialized by CICS/VS prior to entry and must be preserved throughout execution of the program. For ANS COBOL and PL/I, this reservation of registers is resolved by CICS/VS and is of no concern to the application programmer.

Registers 15 through 11 are available to the user and are kept intact when a CICS/VS macro instruction is issued; the contents of register 14 are destroyed whenever a CICS/VS macro instruction is issued.

CICS/VS macro instructions that can be issued to transfer control from or to an application program are listed in the left-hand column of Figure 1-6. The status of all registers upon program entry or upon return to a program is as shown in the remaining columns.

Note: Even though register 14 contains the program entry address, it is not advisable to use register 14 as the base register since it is used by CICS/VS to service requests for CICS/VS supervisory and data management services.

	Registers			
At program entry because of:	15 through 11	12	13	14
Initial Program Entry	Unknown	TCA	CSA	User-program address
LINK	Registers of program issuing the LINK	TCA	CSA	User-program address
XCTL	Registers of program issuing the XCTL	TCA	CSA	User-program address
Following execution of:				
LOAD	Unchanged	TCA	CSA	Next sequential instruction
RETURN (issued by a linked-to program)	Unchanged (from point-of-view program issuing the LINK)	TCA	CSA	Next sequential instruction

Figure 1-6. Register Usage under CICS/VS

RESTRICTIONS

There are certain language considerations that the application programmer should be aware of when writing programs to be run under CICS/VS.

COBOL

The use of CICS/VS macro instructions in an American National Standard (ANS) COBOL application program precludes the use of the following ANS COBOL features:

1. Environment and Data Division entries normally associated with data management services.

2. File Section of the Data Division.

3. Special features: STRING, UNSTRING, SORT, REPORT WRITER, SEGMENTATION, EXHIBIT, TRACE, DISPLAY, and ACCEPT. For CICS/OS/VS, any feature that should require an OS/VS GETMAIN is precluded. (DISPLAY and ACCEPT can be used in conjunction with the system console.)

4. Options that may lead to the issuance of a STXIT (AB) SVC or STAE SVC: FLOW, STATE, STXIT, or SYMDMP for CICS/DOS/VS; FLOW or STATE for CICS/OS/VS.

5. ANS COBOL statements: READ, WRITE, OPEN, CLOSE.

6. QUOTE option, which signifies that literals are to be delineated by quote marks (for example, "74"). Because CICS/VS macro instructions generate COBOL code using apostrophes to delineate literals (for example, '74'), the APOST option must be in effect.

7. The CICS/DOS/VS system does not include support for object code provided by the optimizing option of the ANS COBOL V3 compiler.

In addition, SERVICE RELOAD coding must be added as described under "Additional Guidelines" in Chapter 4 if compiling is accomplished using the ANS COBOL V4 Compiler (5734-CB2), OS/VS COBOL Compiler (5740-CB1), or the DOS/VS COBOL Compiler (5746-CB1) with the optimizing feature.

CICS/VS macro instructions should not be coded within an ANS COBOL statement, since each ANS COBOL statement generated by a CICS/VS macro instruction is terminated by a period.

CICS/VS macro instructions generate COBOL statements which use an apostrophe (') to delineate literals. Code written by the application programmer cannot utilize quotes (") to delineate literals.

Any CICS/VS macro instruction operand which defines a name or label of a storage area or routine should comply with the Assembler language restrictions of eight characters or less. This requirement is a result of preprocessing by the Assembler before COBOL statements are generated.

Any COBOL program that is to run under CICS/VS must contain at least one CICS/VS macro instruction (for example, DFHPC TYPE=RETURN) for proper operation.

PL/I

The use of CICS/VS macro instructions in a PL/I application program precludes the use of the following PL/I features:

3. The PL/I statements: READ, WRITE, GET, PUT, OPEN, CLOSE, DISPLAY, SORT, DELAY, ON, REWRITE, LOCATE, DELETE, REVERT, SIGNAL UNLOCK, STOP, HALT, EXIT, and, for the PL/I Optimizing Compiler, FETCH and RELEASE.

4. The use of floating-point operations; that is, the attribute FLOAT cannot be declared or implied by default. CICS/VS does not save and restore floating-point registers, dump the contents of floating-point registers, or provide for handling of exception conditions that occur during floating-point operations.

The use of CICS/VS macro instructions in a PL/I optimizing compiler application program also precludes the use of the following options:

1. REPORT, FLOW, GONUMBER, GOSTMT.

An application program written in PL/I must consist of an external (MAIN) procedure. Internal procedure CALLs are allowed in a PL/I program to be run under CICS/VS, but external CALLs are not.

Any CICS/VS macro instruction operand which defines a name or label of a storage area or routine should comply with the Assembler language restrictions of eight characters or less. This requirement is a result of preprocessing by the Assembler before PL/I statements are generated.

LINK-EDITING

Separate ANS COBOL routines cannot be link-edited together. Neither can separate PL/I routines. Assembler-language routines may be link-edited, but called routines must conform to CICS/VS application program requirements. Facilities comparable to link-editing are provided under CICS/VS through LINK and XCTL (transfer control) macro instructions, which the application programmer can use to set up communication between programs.

CONSIDERATIONS FOR A VIRTUAL STORAGE ENVIRONMENT

CICS/VS operates in a virtual storage environment. The key objective of programming techniques in a virtual storage system is the reduction of page faults--those cases in which a program refers to an instruction or data that does not reside in real storage. When this occurs, the page in virtual storage that contains the referenced instruction or data must be paged into real storage. The more paging required, the lower the overall system performance.

The application programmer who writes programs to be run in a virtual environment should understand the following concepts:

- locality of reference - the consistent reference, during a program's execution, to instructions and data within a relatively small number of pages (compared to the total number of pages in a program) for relatively long time periods

- working set - the number and combination of pages of a program needed for satisfactory performance (low paging rate) during a given time period

In general, the application programmer should use techniques to improve a program's locality of reference and to minimize the size of its working set at any time during execution of the program. Some guidelines are given below:

1. To achieve locality of reference, processing should be sequential for both code and data, insofar as possible.

   a. Initialize data as close as possible to its first use.

   b. Define new data items as close as possible to the items that use them.

   c. Define arrays or other data structures in the order in which they will be referred to; refer to elements within structures in the order in which they are stored, for example, rowwise rather than columnwise when using PL/I.

   d. Separate error-handling or unusual-situation routines from the main section of a program; they should be subprograms.

   e. Subprograms that are short and used only once or twice (other than those in d above) should be coded inline in the calling program.

2. To achieve minimum use of real storage, the amount of storage that a program refers to in a given time period should be as low as possible. A program should have small working sets.

   a. Write highly modular code and then structure that code according to frequency and anticipated time of reference.

   b. Use separate subprograms whenever the flow of your program suggests that execution will not be sequential.

3. Use techniques to ensure validity of reference, which means to ensure that few storage references retrieve useless data.

   a. Avoid long searches for data.

   b. Use data structures that can be addressed directly, such as arrays, rather than structures that must be searched, such as chains.

   c. Avoid indirect addressing and any methods that simulate indirect addressing.

When all page frames in a real storage environment are filled and another page must be loaded into storage, a page replacement operation is required. The operating system replaces first those pages that have not been referred to for the longest period of time. If a page to be replaced has been modified, that page must be paged out onto virtual storage before the required page can be read in. The more page-out operations required, the lower the overall performance of the system.

To avoid the necessity for page-out operations, the application programmer can code his program so that page-out operations are not required when a page containing a portion of his program must be replaced in real storage. He need only avoid modifying instructions or data within his program. A program in which neither instructions nor data is modified is said to be truly reentrant. As noted earlier, programs to be run in a CICS/VS environment must be quasi-reentrant. For performance reasons, it may be wise to take a further step: to make them truly reentrant programs.

The CICS/VS application programmer should not attempt to use overlays, that is, to incorporate paging techniques within his programs. System paging is automatic and works better anyway.

ASSEMBLY-TIME SERVICE

In addition to knowing the execution-time considerations discussed in this chapter, the application programmer should be aware of an assembly-time (or compile-time) service available under CICS/VS. The DFHCOVER macro instruction can be issued to request that the assembler or compiler in use print a cover page on each of two consecutive pages. This ensures that the application program listing may be torn off with one of the cover pages face up. Pertinent information (program name, date, time of assembly, remarks, and so on) may then be written on the cover page.

The DFHCOVER macro instruction requires no operands and nothing else should appear on the same coding line.

If the DFHCOVER macro instruction is coded as part of an Assembler-language application program, it should be coded as the first instruction in the program. If desired, however, this macro instruction may be coded after anything that is not vital to the listing (such as the TITLE line).

If the DFHCOVER macro instruction is coded as part of an ANS COBOL application program, it should precede the IDENTIFICATION DIVISION statement.

The first card of a PL/I source deck is printed as a header on each page of the source listing. This means that when the DFHCOVER macro instruction is part of a PL/I application program, the first card should be a comments card containing information that the application programmer wants printed as a header. The second card should contain the DFHCOVER macro instruction. The actual PL/I code should begin on the third card of the source deck.

Since column 1 is used by the DFHCOVER macro for line and page spacing under PL/I, column 1 must be defined as reserved for control characters and columns 2-72 must be defined as available for data. This is accomplished through the *PROCESS card for CICS/DOS/VS and the EXEC card for CICS/OS/VS. For further information concerning PL/I compile-time services, see the DOS PL/I Optimizing Compiler Programmer's Guide, OS PL/I (F) Programmer's Guide, and OS PL/I Optimizing Compiler Programmer's Guide.

The examples in Appendix A show how the DFHCOVER macro instruction is used.

CHAPTER 2. STORAGE DEFINITION

SYMBOLIC STORAGE DEFINITIONS

CICS/VS provides symbolic storage definitions (dummy sections) to describe the layouts of control sections of a number of storage areas. These storage definitions are contained in the CICS/VS libraries and can be copied into application programs. When combined with user-defined layouts of the user's sections of the storage areas, they provide symbolic addressing to the storage areas.

The storage areas for which symbolic storage definitions are provided are of three types:

1. Control areas
2. Work areas
3. Input/output areas

Some of the storage areas are statically created by CICS/VS during system initialization, and others are dynamically acquired and released during execution of the system. Some of the areas are acquired or created by CICS/VS; some are acquired directly by the application program; and some are acquired by either CICS/VS or the application program. Figure 2-1 lists the CICS/VS storage areas of interest to the application programmer, indicating which are control areas, which are work areas, and which are I/O areas; it also indicates which are acquired by the user and which are acquired by CICS/VS.

	Control Areas	Work Areas	I/O Areas	Acq'd by User	Acq'd by CICS/VS
Common System Area (CSA)	X	X			X
Common Work Area (CWA)		X			X
Task Control Area (TCA)	X				X
Transaction Work Area (TWA)		X			X
Journal Control Area (JCA)	X			X	X
File Work Area (FWA)		X	X		X
VSAM Work Area (VSWA)		X	X		X
Storage Accounting Area (SAA)		X		X	X
Terminal I/O Area (TIOA)			X	X	X
Transient Data Input Area (TDIA)			X		X
Transient Data Output Area (TDOA)			X	X	
Temporary Storage I/O Area (TSIOA)			X	X	X
File I/O Area (FIOA)			X		X
Terminal Control Table Terminal Entry (TCTTE)	X				X

Figure 2-1. CICS/VS Storage Areas

All CICS/VS storage areas, with the exception of the terminal control table terminal entry (TCTTE), the journal control area (JCA), and VSAM work areas (VSWAs), consist of two logical and unique sections. The control section is used primarily by CICS/VS; the user's section is defined and used exclusively by the application program. This logical division exists whether the storage is statically created (for example, the common system area) or dynamically acquired (for example, a terminal input/output area).

A storage accounting field is built by CICS/VS for every dynamic storage area acquired for the user. Eight bytes at the front and eight bytes at the back of each dynamic storage area are used by CICS/VS for control information. The user should take particular care not to alter or destroy either of these fields. If one of these areas has been altered or destroyed, CICS/VS may be abnormally terminated.

Two ccntrol areas, the common system area (CSA) and the task control area (TCA), must be symbolically defined in every application program; the other control areas (TCTTE and JCA), the work areas, and the I/O areas are selected at the option of the user. It is the user's responsibility to copy symbolic storage definitions into his program for the required control areas as well as for any other storage areas used by his program. (CICS/VS storage areas are summarized in tabular form in Appendix B.)

The identifiers CSA, TCA, and so on in Figure 2-1 are used throughout this manual. They are also used in symbolic names, or labels, within CICS/VS modules and must be used by the application programmer to refer to the data that they represent. Fields within a storage area often begin with the characters of the label for that area. For example, TCA stands for Task Control Area, TCAFCAAA is a field in the TCA that points to a Facility Control Area, TCASCSA is a field in the TCA that points to a Storage Control Storage Area, and so on. The application programmer becomes familiar with these labels through repetitive use. He should assume that when labels of this type appear in this manual they must be used as indicated in application programs. For the reader's convenience, CICS/VS labels in common use are summarized in Appendix C.

Depending on the programming language used, a statement of one of the forms shown below is required to copy a symbolic storage definition into an application program.

1. Assembler-language COPY statement of the form:

```
COPY name
```

2. ANS COBOL COPY statement of the form:

```
01 name COPY name.
```

specified in the Linkage Section of the Data Division.

3. PL/I preprocessor statement of the form:

```
%INCLUDE (name);
    or
%INCLUDE name;
```

For example, assume that one or more terminal input/output areas (TIOAs) are to be acquired during program execution. One of the statements below must be included, depending on whether the program is written in Assembler language, ANS COBOL, or PL/I, respectively.

```
Assembler:      COPY DFHTIOA

ANS COBOL:      01 DFHTIOA COPY DFHTIOA.

PL/I:           %INCLUDE (DFHTIOA);;
```

This statement copies the storage definition as a description or map of a particular area. It does not reserve the storage area. As pointed out above, sometimes CICS/VS acquires the area; in other cases, the user acquires it. In either situation, the application programmer must effectively map the storage definition that he has copied into his program over the storage area acquired. He does so by moving the address of the area (stored in a particular location by CICS/VS) into what effectively becomes a base locator for that area. Addressability through this base locator is limited to 4096 (0 through 4095) bytes for any CICS/VS provided storage definition. Depending on the programming language, a statement of one of the following forms must be used to establish addressability to the area:

1. Assembler-language statement of the form:

   L base-locator, location-containing-address

2. ANS COBOL statement of the form:

   MOVE location-containing-address TO base-locator.

3. PL/I based pointer variable of the form:

   base-locator = location-containing-address;

For example, assume that a terminal input/output area (TIOA) has been acquired during program execution. TCASCSA is a four-byte field in the TCA that contains the address of the dynamic storage area that was acquired. TIOABAR is the TICA base address register. One of the statements below must be executed, depending on whether the program is written in Assembler language, ANS COBOL, or PL/I, respectively.

```
Assembler:      L TIOABAR,TCASCSA

ANS COBOL:      MOVE TCASCSA TO TIOABAR.

PL/I:           TIOABAR=TCASCSA;
```

Figure 2-2 contains the names used in copying CICS/VS-provided symbolic storage definitions of control sections into an application program and the names that represent base addresses used in establishing addressability. (See also Appendix B.)

CICS/VS Storage Area	Abbreviation	Symbolic Names For Defined Storage	Base Locator Or Base Address Register	Assembler-Language General-Purpose Register Assignment
Common System Area	CSA	DFHCSADS	CSACBAR	13
Common Work Area	CWA	User-defined	CSACBAR	13
Task Control Area	TCA	DFHTCADS	TCACBAR	12
Transaction Work Area	TWA	User-defined	TCACBAR	12
File Work Area	FWA	DFHFWADS	FWACBAR	*
Storage Accounting Area	SAA	DFHSAADS	SAACBAR	*
Terminal Input/Output Area	TIOA	DFHTIOA	TIOABAR	*
File Input/Output Area	FIOA	DFHFIOA	FIOABAR	*
Transient Data Input Area	TDIA	DFHTDIA	TDIABAR	*
Transient Data Output Area	TDOA	DFHTDOA	TDOABAR	*
Temporary Storage Input/Output Area	TSIOA	DFHTSIOA	TSIOABAR	*
Journal Control Area	JCA	DFHJCADS	JCABAR	*
VSAM Work Area	VSWA	DFHVSWA	VSWABAR	*
Terminal Control Table Terminal Entry	TCTTE	DFHTCTTE	TCTTEAR	*
Application Program Storage	-	-	User-defined	*

*Any register except 12, 13, and 14 which are utilized by CICS/VS, and 0 which cannot be used as a base or an index register.

Figure 2-2. Symbolic Names and Base Addresses of CICS/VS Storage Areas

All storage acquired by the application program through CICS/VS storage management is controlled by a technique that chains together all storage associated with a particular transaction. This feature allows CICS/VS to release all storage associated with a transaction, either upon request from the user or when the transaction is terminated, normally or abnormally.

The common system area (CSA) is the head of the chain. Its address is provided by CICS/VS. The CSA points to the task control area (TCA) which in turn points to several other storage areas. Figure 2-3 illustrates the chaining of CICS/VS storage areas and indicates the symbolic base address used to locate each storage area.

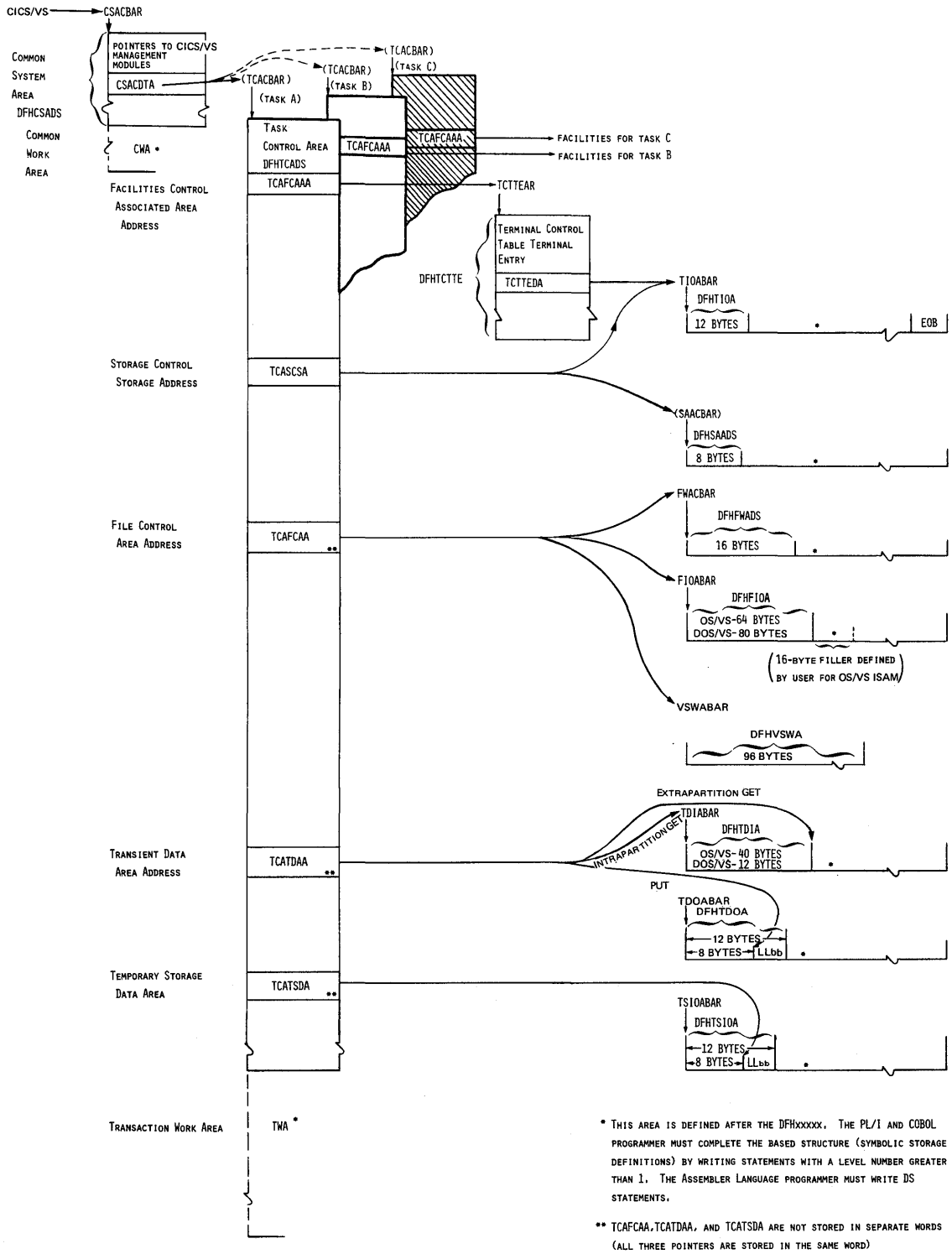


Figure 2-3. Chaining of CICS/VS Storage Areas

REQUIRED STORAGE AREAS

At least two storage area definitions, namely, those which define the CSA and the TCA, are required in every application program to be run under CICS/VS. The following sections describe these areas. The fields of special significance for the application programmer are discussed in detail. Services performed by CICS/VS components (recall the list of components in the preceding section of this manual) are mentioned as necessary. Some tables that are basic to CICS/VS operation are also mentioned. These tables are explained in greater detail in the CICS/VS System Programmer's Reference Manual.

COMMON SYSTEM AREA (CSA)

The CSA is an area of static storage that contains areas and data required for the operation of CICS/VS. It also contains a user-defined common work area (CWA) that can be referred to by application programs.

Data in the CSA that is required for operation of CICS/VS includes:

- CICS/VS save areas
- Addresses of CICS/VS management programs
- Control system and user statistics accumulators
- Addresses of CICS/VS system control tables
- Common system constants
- System control parameters

The fields of the CSA that are of particular significance to the application programmer are as follows:

CSACTODB: This four-byte binary field contains the time of day in hundredths of a second. The time of day is updated periodically during task dispatching and reset at midnight; its accuracy depends on the task mix and the frequency of task switching.

CSATODP: This four-byte packed decimal field contains the time of day in the form HHMMSSt+, that is, expressed to tenths of a second. The time of day is updated periodically during task dispatching and reset at midnight; its accuracy depends on the task mix and the frequency of task switching.

CSAJYDP: This four-byte packed decimal field contains the Julian date in the form YYDDD+. (The left-most byte is binary zeros.) The value is changed at midnight.

CSAWABA: This field represents the beginning of the common work area (CWA) and ensures doubleword storage alignment for it.

CSAOPFLA: This four-byte field contains the address of the CSA optional features list. This list contains various pointers used by the journal control program, the built-in functions program, and so on. The field need not be referred to explicitly in an Assembler-language or PL/I application program but must be used to establish addressability for the CSA optional features list in an ANS COBOL application program.

Common Work Area (CWA)

The CWA is an area within the CSA that can be used by application programs for the retention of temporary data, the accumulation of statistics, the passing of parameters, and so on. The size of this area is determined by the user installation at system generation. It is initially set to binary zeros. Its contents can be accessed and altered by any number of tasks during CICS/VS operation.

Addressability for the CWA is provided when copying the CICS/VS storage definition for the CSA. However, addressability is limited to a combined total of 4096 (0 through 4095) bytes for the CSA and CWA. Addressability for any portion of the CWA extending beyond the 4095-byte limit is the responsibility of the user.

Since the CWA is available to any task while it has control of the system, it is not advisable for an application program to indiscriminately use this area for retention of data when requesting CICS/VS services. Another transaction may get control and destroy the data. However, if the user designs his application programs to expect and maintain a common, user-established format within the CWA, there is no reason why the area cannot be shared by several tasks. For example, a statistics accumulator within the CWA can be updated by more than one transaction.

TASK CONTROL AREA (TCA)

The TCA is an area of main storage acquired dynamically by CICS/VS when the task (transaction) is originated by task control. Once acquired, the TCA exists until the task is terminated. It is used to represent the current status of the task and its relative dispatching priority. During execution of the task, the user has the capability of changing the priority through task management services; further processing of the task is scheduled accordingly.

The TCA provides the following items for its associated task:

- Register save areas
- Unique fields (parameter areas) for communicating requests to CICS/VS
- Address of the related Facility Control Area (FCA)
- Task storage chain addresses

The TCA provides no space for any residual data such as statistics. However, the TCA can be extended to include a transaction work area (TWA), the size of which is determined by the user to meet the needs of the transaction. (See "Transaction Work Area.")

The TCA consists of three logical sections:

- CICS/VS system control section
- Communication section
- Transaction Work Area (optional)

The CICS/VS system control section contains control addresses and data needed by CICS/VS to control the task. Access to this section is limited to CICS/VS management programs, CICS/VS service programs, and user-written service programs.

The communication section is used by CICS/VS and by user-written application programs for communication between the task and CICS/VS management programs and service programs.

The optional transaction work area is reserved for the exclusive use of the task.

In those cases in which a task is initiated from a terminal (nearly always the case), CICS/VS places into the TCA the address of the terminal control table terminal entry (TCTTE) associated with the terminal. The TCTTE, in turn, contains the address of the terminal input/output area (TIOA).

The fields of the TCA that are of particular significance to the application programmer are as follows:

TCAFCAAA: This four-byte field contains the address of the facility control area associated with the facility that initiated the transaction. This field can contain the address of a terminal control table terminal entry, a destination control table entry, or an automatic task initiator control area. This field can also be used to pass the address of a user-defined area whenever one task attaches another nonterminal-oriented task. It can contain any meaningful communication between two asynchronous tasks. When used for such purposes, the high-order byte of the field must contain zeros.

If the user's application program is to communicate with the terminal, TCAFCAAA must contain the address of the appropriate terminal control table terminal entry (TCTTE). This allows the application program to refer to any data in the TCTTE.

(The following fields are contained in the Common Control Communication Area of the TCA. This means that the contents of these fields will not necessarily be maintained during any CICS/VS service request.)

TCAPCPI: This eight-byte field contains the identification of the requested program. The program identification is left-justified and must meet label requirements of the operating system in use. The CICS/VS processing program table (PPT) must also contain this program identification.

This field (TCAPCPI) can be filled prior to issuing a program control (DFHPC) macro instruction. (See "Program Services" in Chapter 6.) If the application program places the program identification in TCAPCPI prior to the execution of the macro instruction, the PROGRAM=name operand should be omitted from the macro instruction. For example, the program identification can be placed in TCAPCPI prior to issuing a DFHPC TYPE=LINK macro instruction when an application program is testing to determine to which program to link. On the basis of the test, the application program should place the program identification in TCAPCPI and then execute a DFHPC TYPE=LINK macro instruction without the PROGRAM=name operand. With this technique, one macro instruction can be issued repetitively to link to many different programs.

TCAPCAC: This four-byte field contains the termination code for the DFHPC TYPE=ABEND macro instruction. The termination code must be left-justified and must be the user's termination code.

This field (TCAPCAC) can be filled by the application program prior to issuing the DFHPC TYPE=ABEND macro instruction; in this case, the ABCODE=YES operand should be coded if a dump is requested. Generally, the application program places the termination code in TCAPCAC prior to the execution of the DFHPC TYPE=ABEND macro instruction when testing to determine which type of termination is desired.

TCASCSA: This four-byte field contains the address of the storage obtained after the execution of a DFHSC TYPE=GETMAIN macro instruction; it must also contain the address of the storage to be released prior to execution of a DFHSC TYPE=FREEMAIN macro instruction. (See "Storage Services" in Chapter 6.) The application programmer must remember that the first eight bytes at this address are always the storage accounting field used by CICS/VS storage management. Care should be taken never to alter the contents of a storage accounting field.

The address of the storage obtained from a DFHSC TYPE=GETMAIN macro instruction is automatically placed in TCASCSA except when a conditional GETMAIN request (COND=YES) has been issued and storage is not available. In this case, CICS/VS storage management places binary zeros in this field and returns control to the application program. The application program should specify a symbolic base address for the storage area obtained and move the storage address returned in TCASCSA to this symbolic base address.

TCADCNB: This two-byte field contains the length (in bytes) of the main storage area to be dumped by CICS/VS dump management. (See "Dump Services" in Chapter 8.) The application program can place a hexadecimal representation of the number of bytes requested into this field prior to execution of the DFHDC TYPE=PARTIAL macro instruction.

TCASCNB: This two-byte field contains the number of bytes of storage to be obtained by CICS/VS storage management. This field can be filled by the application program with a hexadecimal representation of the number of bytes requested prior to execution of the DFHSC TYPE=GETMAIN macro instruction. If the application program places a value in this field prior to execution of a DFHSC TYPE=GETMAIN macro instruction, the NUMBYTE=value operand must be omitted. When the storage is obtained, TCASCNB is overlaid with a portion of the address of the storage obtained.

TCASCIB: This one-byte field contains the bit configuration used for the initialization of dynamically acquired storage. The field can be filled by the application program with the desired bit configuration prior to execution of a DFHSC TYPE=GETMAIN macro instruction, in which case the INITIMG=YES operand must be coded.

TCAFCDI: This eight-byte field contains the symbolic data set identification for the data set to which a record is to be written or from which a record is to be retrieved. This identification must correspond exactly with the identification of the required data set placed in the file control table by the system programmer at system generation. It must be left-justified in TCAFCDI. The application program can place the data set identification in this field prior to execution of a file control DFHFC TYPE=GET or TYPE=SETL macro instruction. (See "File Services" in Chapter

6.) If this field is filled prior to execution of the DFHFC macro instruction, the DATASET=name operand must be omitted.

TCAFCRI: This four-byte field contains the address of the user's record identification field when making a request for CICS/VS file management services. The application program can place the address in this field prior to execution of a DFHFC TYPE=GET, DFHFC TYPE=PUT,TYPOPER=NEWREC, or DFHFC TYPE=SETL macro instruction. The RDIDADR=symbol operand should be omitted if TCAFCRI is filled prior to execution of the macro instruction.

TCAFCSI: This eight-byte field contains the symbolic segment set identification. This identification must match the identification of the requested segment set placed in the file control table by the system programmer at system generation. It must be left-justified in TCAFCSI.

The application program can place the segment set identification in this field prior to issuing the DFHFC TYPE=GET, DFHFC TYPE=PUT, DFHFC TYPE=SETL, or DFHFC TYPE=GETNEXT macro instruction. If this field is filled prior to execution of the DFHFC TYPE=GET or DFHFC TYPE=PUT macro instruction, the SEGSET=YES operand must be coded as part of the macro instruction.

TCAFCAI: This eight-byte field contains the symbolic identification of the first index data set to be searched in an indirect accessing hierarchy. The application program can place the desired indirect access identification (as previously established in the file control table by the system programmer at system generation time) in the field prior to execution of a DFHFC TYPE=GET macro instruction. When the application program places the identification in TCAFCAI, it must be left-justified and the INDEX=YES operand must be coded as part of the macro instruction.

TCAFCAA: This four-byte field contains the address of the file input/output area (FIOA), file work area (FWA), or VSAM work area (VSWA).

TCAFCNRD: This two-byte field contains a count of the number of records deleted upon return to the application program after completion of a DFHFC TYPE=DELETE macro instruction in which a generic key is specified.

TCAFCURL: This two-byte field contains the length of an undefined record being written to a DAM data set. The value must be placed in this field by the application program prior to issuing the DFHFC TYPE=PUT,TYPOPER=NEWREC macro instruction that requests the write of the undefined record.

TCAFCTR: This one-byte field contains the type of file control request/response. Request codes are set by issuing the DFHFC macro instruction. Responses are automatically placed in TCAFCTR by file management after completion of the event requested.

TCATDTR: This one-byte field contains the type of transient data control request/response. (See "Transient Data Services" in Chapter 6.) Request codes are set by issuing the DFHTD macro instruction. Responses are automatically placed in TCATDTR by transient data management after completion of a transient data event.

TCATSTR: This one-byte field contains the temporary storage control request/response. Request codes are set by issuing the temporary storage macro instruction (DFHTS). Responses are automatically placed in TCATSTR by temporary storage management after completion of a temporary storage event.

TCAICTR: This one-byte field contains the interval control request/response. (See "Time Services" in Chapter 6.) Request codes are set by issuing the interval control macro instruction (DFHIC). Responses are automatically placed in TCAICTR by time management after completion of an interval control service request.

Transaction Work Area (TWA)

The TWA is an extension of the TCA and is created at the option of the user to provide a work area for a given transaction (task). The TWA can be used for the accumulation of data and intermediate results during the execution of the transaction. It can also be used when the amount of working storage for a transaction is relatively static, when data must be passed between user-written application programs, or when data must be accessed by different programs during transaction processing. During multiple entries of data for a transaction, the application programs might retain the data in TWA. This approach cannot be used for multiple transactions; the TWA is released automatically at task termination.

The size of the TWA for the transaction must be specified in the program control table by the system programmer at system generation time. The application programmer should work with the system programmer in determining the amount of storage required. He must then define the storage area immediately following the definition of the TCA in his application program. The size of TWAs within the system can vary according to specific transaction needs. (For a discussion of establishing the TWA, see the explanation of the program control table in the CICS/VS System Programmer's Reference Manual.)

Addressability of the TWA is provided when copying the CICS/VS storage definition for the TCA. However, addressability is limited to a combined total of 4096 (0 through 4095) bytes for the TCA and TWA. Addressability for any portion of the TWA extending beyond the 4095-byte limit is the responsibility of the user.

CHAPTER 3. COPYING STORAGE DEFINITIONS - ASSEMBLER LANGUAGE

The Assembler-language programmer must define storage for the CICS/VS control areas and any other storage areas required for the processing of his program. He accomplishes this by using the Assembler-language COPY statement to (1) copy the appropriate symbolic storage definitions into his program and (2) specify the names of the storage areas being defined. All registers are at his disposal, except registers 12, 13, and 14 (which are used by CICS/VS).

All programs must contain statements to copy the symbolic storage definitions for the common system area (CSA) and the task control area (TCA). CICS/VS macro expansions resulting from macro instructions that the application programmer uses refer to fields within these areas, so their locations must be identified. Whether additional definitions must be copied depends on the processing requirements (storage areas and macro instructions used) of the application program.

STATIC STORAGE DEFINITION

During CICS/VS initialization, the CSA is statically allocated as part of the CICS/VS nucleus. For each terminal with which communication is to occur, the terminal control table terminal entry (TCTTE) is included in the statically allocated terminal control table (TCT). These constitute the static storage areas for which COPY statements may be required.

COMMON SYSTEM AREA (CSA)

The statement

```
COPY DFHCSADS
```

copies the symbolic storage definitions for the CSA and the CSA optional features list (CSAOPL) and assigns register 13 as the base register.

If CICS/VS was generated to support a common work area (CWA) within the CSA, the application programmer may wish to include his own symbolic definition for that area following the COPY DFHCSADS statement. For example:

```
          COPY   DFHCSADS
BUCKET1   DS     F
BUCKET2   DS     F
TEMPNAME  DS     CL8
```

TERMINAL CONTROL TABLE TERMINAL ENTRY (TCTTE)

The statement

```
COPY DFHTCTTE
```

copies the symbolic storage definition for the TCTTE. This symbolic storage definition is necessary when the user desires to obtain the address of the current terminal I/O area (the current terminal control table terminal entry data address, or TCTTEDA) or to request a terminal control service via the DFHTC macro instruction. The user must code an EQU statement to set up a base register for the TCTTE, equating the

label TCTTEAR to a general-purpose register. He must also establish addressability for the TCTTE by loading the address at TCAFCAAA into TCTTEAR. The following is an example of the coding required:

```
TCTTEAR EQU  5
        COPY DFHTCTTE
        .
        .
        .
        L    TCTTEAR,TCAFCAAA
```

DYNAMIC STORAGE DEFINITION

During initiation and execution of a transaction (task), the TCA, TIOA, and other storage areas required by the transaction are dynamically allocated by CICS/VS storage management, upon request from either the application program or a CICS/VS management function. The application programmer must provide symbolic storage definitions for these storage areas by using COPY statements as described below.

TASK CONTROL AREA (TCA)

The statement

```
COPY DFHTCADS
```

copies the symbolic storage definition for the TCA (excluding the CICS/VS control section) and assigns register 12 as the base register. If the user's application program uses a transaction work area (TWA), DS statements for that storage area must immediately follow the COPY statement. The following is an example of the coding required to symbolically define storage for both the TCA and TWA:

```
        COPY DFHTCADS
NAME    DS   CL20
STREET  DS   CL20
CITY    DS   CL10
STATE   DS   CL3
```

If it is necessary for the Assembler-language programmer to access the CICS/VS system control section of the TCA, a copy of the symbolic storage definition for the entire TCA can be obtained by using the CICS/VS macro instruction

```
DFHTCA CICSYST=YES
```

in place of the statement COPY DFHTCADS. Addressability to the communication section of the TCA and to the transaction work area (TWA) is provided automatically by CICS/VS through register 12. Addressability to the CICS/VS system control section must be provided by the application programmer; for example:

```
L     WRKREG,TCASYAA
USING DFHSYTCA,WRKREG
.
.
.
DROP  WRKREG
```

TERMINAL INPUT/OUTPUT AREA (TIOA)

The statement

```
COPY DFHTIOA
```

copies the symbolic storage definition for the CICS/VS control section of the TIOA. This storage definition should precede the user's definition of a terminal input or output message. The user must code an EQU statement to set up a base register for the TIOA, equating the label TIOABAR to a general-purpose register. He can then request an action that requires a TIOA. For example, he can issue a DFHSC TYPE=GETMAIN macro instruction requesting CICS/VS storage control to obtain dynamic storage for a TIOA for his program, as illustrated here:

```
TIOABAR EQU   9
        COPY  DFHTIOA
NAME    DS    CL20
STREET  DS    CL20
        DS    CL5
        .
        .
        .
        DFHSC TYPE=GETMAIN,NUMBYTE=XXX,CLASS=TERMINAL
        L     TIOABAR,TCASCSA
```

For additional information about GETMAIN, see "Obtain and Initialize Main Storage (GETMAIN)" under "Storage Services" in Chapter 6.

FILE INPUT/OUTPUT AREA (FIOA)

The statement

```
COPY DFHFIOA
```

copies the symbolic storage definition for the CICS/VS control section of the FIOA. This storage definition should precede the user's defined layout of a file input or output area when reading an unblocked record without updating or segmenting, or when reading DAM blocked records without deblocking. If desired, the user can identify the area returned in response to a user file request as a FIOA, rather than a FWA or VSWA, by testing FIOAM for a mixed condition at label FIOAIND. The user must code an EQU statement to set up a base register for the FIOA, equating the label FIOABAR to a general-purpose register. The FIOA is automatically acquired by CICS/VS file management whenever a request is made by the user to access a data base data set. If data is retrieved using the Indexed Sequential Access Method (ISAM) under CICS/OS/VS, a 16-byte filler must be defined prior to the user's data definition. The user must establish addressability for an FIOA acquired in response to a DFHFC macro instruction before referring to the FIOA. See the example below which also shows the optional test for FIOA identification.

```
FIOABAR EQU   7
        COPY  DFHFIOA
        DS    16X               OS/VS ISAM FILLER
NAME    DS    CL20
STREET  DS    CL5
        .
        .
        .
        L     FIOABAR,TCAFCAA
        TM    FIOAIND,FIOAM     WAS A FIOA RETURNED?
        BM    GOTFIOA           YES
```

FILE WORK AREA (FWA)

The statement

```
COPY DFHFWADS
```

copies the symbolic storage definition for the CICS/VS control section of the FWA. This storage definition should precede the user's defined layout of a file record area when reading or updating an existing blocked or segmented record, when adding a new record to a file, or when retrieving records using the browse feature. If desired, the user can identify the area returned in response to a user file request as a FWA, rather than a FIOA or VSWA, by testing FWAM for a ones condition at label FWAIND. The user must code an EQU statement to set up a base register for the FWA, equating the label FWACBAR to a general-purpose register. He must also establish addressability for an FWA acquired in response to a DFHFC macro instruction prior to any reference to the FWA. The following example illustrates the coding required, as well as the optional test for FWA identification:

```
FWACBAR EQU   7
        COPY  DFHFWADS
NAME    DS    CL20
STREET  DS    CL30
ZIPCODE DS    CL5
        .
        .
        .
        L     FWACBAR,TCAFCAA
        TM    FWAIND,FWAM       WAS FWA RETURNED?
        BO    GOTFWA            YES
```

VSAM WORK AREA (VSWA)

The statement

```
COPY DFHVSWA
```

copies the symbolic storage definition for the CICS/VS control section of the VSAM work area (VSWA) and must be present in all programs using locate mode I/O. (See "File Services" in Chapter 6.) If desired, the user can identify the area returned in response to a user file request as a VSWA, rather than a FIOA or FWA, by testing VSWAM for a zero condition at label VSWAID. The user must code an EQU statement to set up a base register for the VSWA, equating the label VSWABAR to a general-purpose register. After a VSWA is acquired by CICS/VS in response to a DFHFC macro instruction utilizing locate mode I/O, the user must establish addressability for the VSWA prior to any reference to that area. The following example illustrates the coding required, as well as the optional test for VSWA identification:

```
        EQU   7
        COPY  DFHVSWA
        .
        .
        .
        L     VSWABAR,TCAFCAA
        TM    VSWAID,VSWAM      WAS VSWA RETURNED?
        BZ    GOTVSWA           YES
```

TRANSIENT DATA INPUT AREA (TDIA)

The statement

```
COPY DFHTDIA
```

copies the symbolic storage definition for the CICS/VS control section of the intrapartition TDIA. This storage definition should precede the user's defined layout of the message area used for data obtained from an intrapartition destination by means of a DFHTD TYPE=GET macro instruction. (See "Acquire Queued Data (GET)" under "Transient Data Services" in Chapter 6.) The user must code an EQU statement to set up a base register for the TDIA, equating the label TDIABAR to a general-purpose register. He must also establish addressability for the TDIA following a DFHTD macro instruction. The following is an example of the coding required:

```
TDIABAR EQU   9
        COPY  DFHTDIA
NAME    DS    CL20
STREET  DS    CL20
        .
        .
        .
        L     TDIABAR,TCATDAA
```

TRANSIENT DATA OUTPUT AREA (TDOA)

The statement

```
COPY DFHTDOA
```

copies the symbolic storage definition for the CICS/VS control section of the intrapartition TDOA. For consistent documentation of the application program, this storage definition should precede the user's defined layout of the message area for transient data to be directed to an intrapartition or extrapartition destination by means of a DFHTD TYPE=PUT macro instruction. (See "Dispose of Data (PUT)" under "Transient Data Services.") The user must code an EQU statement to set up a base register for the TDOA, equating the label TDOABAR to a general-purpose register. The address of the length field labeled TDOAVRL must be given to transient data control either through the TDADDR operand of the DFHTD macro instruction or by placing it in TCATDAA. The following is an example of the coding required:

```
TDOABAR EQU   9
        COPY  DFHTDOA
TIME    DS    CL4
DATE    DS    PL3
INTERM  DS    CL4
OUTTERM DS    CL4
        .
        .
        .
        DFHSC TYPE=GETMAIN,CLASS=TRANSDATA,NUMBYTE=XX
        L     TDOABAR,TCASCSA
        .
        .
        .
        DFHTD TYPE=PUT,DESTID=POST,TDADDR=TDOAVRL
```

TEMPORARY STORAGE INPUT/OUTPUT AREA (TSIOA)

The statement

```
COPY DFHTSIOA
```

copies the symbolic storage definition for the CICS/VS control section of the TSIOA. This storage definition should precede the user's defined layout of input/output work areas for temporary storage. The user must code an EQU statement to set up a base register for the TSIOA, equating the label TSIOABAR to a general-purpose register. The address of the length field labeled TSIOAVRI must be given to temporary storage control either through the TSDADDR operand of the DFHTS macro instruction or by placing it in TCATSDA. The following is an example of the coding required:

```
TSIOABAR EQU   6
         COPY  DFHTSICA
PAGENO   DS    PL2
TITLE    DS    CL30
LINE1    DS    CL70
         .
         .
         .
         DFHTS TYPE=GET
         L     TSIOABAR,TCATSDA
         SH    TSIOABAR,=H'8'
```

Upon execution of the DFHTS TYPE=GET instruction above, CICS/VS returns the data portion (LLbb field) of the address of the obtained storage area in TCATSDA. To establish addressability to the TSIOA (that is, to use the DFHTSIOA DSECT), the application programmer must subtract eight from this address to point to the storage accounting field of the storage area acquired by CICS/VS. If the TSDADDR operand is included in the DFHTS TYPE=GET macro instruction, this is not required.

STORAGE ACCOUNTING AREA (SAA)

The statement

```
COPY DFHSAADS
```

copies the symbolic storage definition for the SAA. This storage definition should precede the user's defined laycut of a unique work area that he uses within his application program. The user must code an EQU statement to set up a base register for the SAA, equating the label SAACBAR to a general-purpose register. The following is an example of the coding required:

```
SAACBAR  EQU   9
         COPY  DFHSAADS
SYMBLA   EQU   *
NAME     DS    CL50
STREET   DS    CL15
SYMBLB   EQU   *-SYMBIA
         .
         .
         .
         DFHSC TYPE=GETMAIN,INITIMG=00,NUMBYTE=SYMBLB,
               CLASS=USER
         .
         .
         .
         L     SAACBAR,TCASCSA
```

.
.
.

Having copied the symbolic storage definition for the SAA, the application programmer can write a DFHSC TYPE=GETMAIN instruction requesting CICS/VS storage control to obtain main storage for use by his program. He should move the address returned by CICS/VS in TCASCSA to SAACBAR, the base address register for the SAA.

JOURNAL CONTROL AREA (JCA)

The statement

```
COPY DFHJCADS
```

copies the symbolic storage definition for the CICS/VS control section of the journal ccntrol area (JCA) and must be present in all programs requesting journal services. (See "Journal Services" in Chapter 6.) The user must code an EQU statement to set up a base register for the JCA, equating the label JCABAR to a general-purpose register. The following is an example of the coding required:

```
JCABAR   EQU   9
         COPY  DFHJCADS
```

A JCA is acquired dynamically by means of a DFHJC TYPE=GETJCA macro instruction. Addressability to the JCA is automatically provided through the macro expansion, which loads the JCA address into JCABAR.

EXAMPLE OF CICS/VS ASSEMBLER-LANGUAGE APPLICATION PROGRAM

Figure 3-1 is an Assembler-language program written to run under CICS/VS. The program asks a question of the terminal operator, receives a reply, dynamically acquires some storage, and sends the operator's message back to the terminal. In effect, an echo test is performed. (The line numbers in the figure are not part of the program.)

A discussion of the significance of each of the lines of Figure 3-1 follows.

Line Number	Description
01	Assigns base register for program.
02-03	Assigns base registers for TCTTE and TIOA symbolic storage definitions.
04-05	Copies CSA and TCA symbolic storage definitions.
06-07	Defines fields in TWA as save areas to provide for quasi-reentrance.
08-09	Copies TCTTE and TIOA symbolic storage definitions.
10	Defines message area in TIOA.
11-13	Begins program; establishes addressability for program.
14	Establishes addresssability for TCTTE.
15	Establishes addressability for TIOA.
16	Moves message to output area of TIOA.
17	Moves length of message to data length field of TIOA.
18	CICS/VS macro instruction that writes message

	to terminal, waits for operator's reply, and reads operator's reply.
19	Establishes addressability for new TIOA, using address in TCTTE.
20-21	Saves the message and the length of the message in the TWA save areas.
22-25	CICS/VS macro instruction that requests 32 bytes of terminal type storage initialized to blanks.
26	Establishes addressability for new TIOA (address of newly acquired storage area is in TCASCSA field of the TCA).
27	Places address of new TIOA in TCTTE.
28	Moves the message from TWA save area to new TIOA.
29	Moves the message length to data length field of new TIOA.
30	CICS/VS macro instruction that writes message to terminal.
31	CICS/VS macro instruction that returns control to CICS/VS and terminates this task.
32-33	Required for Assembler language.

```
01  BASEREG EQU     2
02  TCTTEAR EQU     11
03  TIOABAR EQU     10
04          COPY    DFHCSADS
05          COPY    DFHTCADS
06  LENGTH  DS      H
07  MESSAGE DS      CL32
08          COPY    DFHTCTTE
09          COPY    DFHTICA
10  MESSG   DS      CL32
11          CSECT
12          BALR    BASEREG,0
13          USING   *,BASEREG
14          L       TCTTEAR,TCAFCAAA
15          L       TIOABAR,TCTTEDA
16          MVC     MESSG,=C'WHAT LANGUAGE AM I CODED IN'
17          MVC     TIOATDL,=H'27'
18          DFHTC   TYPE=(WRITE,READ,WAIT)
19          L       TIOABAR,TCTTEDA
20          MVC     LENGTH,TIOATDL
21          MVC     MESSAGE,MESSG
22          DFHSC   TYPE=GETMAIN,                                  *
23                  CLASS=TERMINAL,                                *
24                  INITIMG=40,                                    *
25                  NUMBYTE=32
26          L       TIOABAR,TCASCSA
27          ST      TIOABAR,TCTTEDA
28          MVC     MESSG,MESSAGE
29          MVC     TIOATDL,LENGTH
30          DFHTC   TYPE=WRITE
31          DFHPC   TYPE=RETURN
32          LTORG
33          END
```

Figure 3-1. Example of CICS/VS Assembler-Language Application Program

CHAPTER 4. COPYING STORAGE DEFINITIONS - ANS COBOL

The application programmer who uses American National Standard (ANS) COBOL must define storage for the CICS/VS control areas and any other storage areas required for the processing of his program. He accomplishes this by using (1) the COPY statement in the Linkage Section of the Data Division to copy the symbolic storage definitions into his program and specify the names of the storage areas being defined, and (2) the MOVE statement in the Procedure Division to establish addressability by moving symbolic storage addresses from one location to another.

The working storage section of an ANS COBOL program should contain only data constants. Variable data should be placed in the CICS/VS transaction work area (TWA) or in an area of dynamic storage acquired by a DFHSC TYPE=GETMAIN macro instruction. (See "Obtain and Initialize Main Storage (GETMAIN)" under "Storage Services" in Chapter 6 to learn more about this capability.)

The statement

```
01 DFHBLLDS COPY DFHBILDS.
```

must be the first statement in the linkage section of the Data Division of an ANS COBOL program that is run under CICS/VS. This statement copies the symbolic storage definition for the linkage section base locator (BLL), which provides the means by which an ANS COBOL program can request dynamically acquired CICS/VS storage areas. Included in this definition are the symbolic base addresses for the common system area (CSA), common system area optional features list (CSAOPL), and task control area (TCA). Symbolic storage definitions for these areas must be copied into every ANS COBOL program.

If the ANS COBOL programmer desires to use CICS/VS storage areas in addition to the CSA and TCA, the COPY statement for the BLL must be followed immediately by statements of the form

```
02 name PICTURE S9(8) USAGE IS COMPUTATIONAL.
```

where name is the symbolic base address used to locate a specific storage area. There must be one of these statements for each additional type of storage needed by the application program. Furthermore, these 02-level statements must be coded in the same order as the corresponding 01-level COPY statements coded subsequently to copy the symbolic storage definitions for the areas into the application program.

If the user is going to communicate with the system by means of a terminal, he needs a terminal input/output area (TIOA) and a terminal control table terminal entry (TCTTE). Assuming that only the required control areas (CSA and TCA), TIOA, and TCTTE are needed for a particular application, the following example shows coding required in the linkage section of the Data Division:

```
01 DFHBLLDS COPY DFHBLLDS.
   02 TCTTEAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   02 TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
01 DFHCSADS COPY DFHCSADS.
01 DFHTCADS COPY DFHTCADS.
01 DFHTCTTE COPY DFHTCTTE.
01 DFHTIOA COPY DFHTICA.
```

STATIC STORAGE DEFINITION

During CICS/VS initialization, the common system area (CSA) is statically allocated as part of the CICS/VS nucleus. For each terminal with which communication is to occur, the terminal control table terminal entry (TCTTE) is included in the statically allocated terminal control table (TCT). The ANS COBOL programmer must provide symbolic storage definitions for the CSA and TCTTE (if needed) as follows.

COMMON SYSTEM AREA (CSA)

The statement

```
01 DFHCSADS COPY DFHCSADS.
```

copies the symbolic storage definition for the CSA. Addressability for the CSA is included.

If CICS/VS was generated to support a common work area (CWA) within the CSA, immediately following the COPY statement in the linkage section the application programmer can define the record layout of the CWA. The following is an example of the coding required:

```
01 DFHCSADS COPY DFHCSADS.
   02 CWA.
      03 FIELD1 PICTURE X(4).
      .
      .
      .
```

TERMINAL CONTROL TABLE TERMINAL ENTRY (TCTTE)

The statement

```
01 DFHTCTTE COPY DFHTCTTE.
```

copies the symbolic storage definition for the TCTTE and must be present in all programs requesting communication with a terminal. The user must code the statement

```
MOVE TCAFCAAA TO TCTTEAR.
```

in the appropriate place in the Procedure Division to establish addressability for the TCTTE. TCAFCAAA contains the address of the facility that initiated the transaction. TCTTEAR is the terminal control table terminal entry address register.

DYNAMIC STORAGE DEFINITION

During initiation and execution of a transaction (task), the task control area (TCA), the terminal input/output area (TIOA), and other storage areas required by the transaction are dynamically allocated by CICS/VS storage management, upon request from either the application program or a CICS/VS management function. The ANS COBOL programmer must provide symbolic storage definitions for these storage areas as follows.

TASK CONTROL AREA (TCA)

The statement

```
01 DFHTCADS COPY DFHTCADS.
```

copies the symbolic storage definitions for the CSA optional features list and the TCA. The user must code the statements

```
MOVE CSACDTA TO TCACBAR.
MOVE CSAOPFLA TO CSAOPBAR.
```

at the appropriate place in the Procedure Division to establish addressability for the TCA and the CSA optional features list. CSACDTA contains the address of the storage area obtained for the TCA (the common system area currently dispatched task address). This address is stored in TCACBAR, the TCA control base address register.

If the user desires to append a transaction work area (TWA) to the TCA, immediately following the COPY statement in the Linkage Section he must define the record layout of the TWA. The following is an example of the coding required:

```
01 DFHTCADS COPY DFHTCADS.
   02 TWA PICTURE X(40).
    .
    .
    .
```

TERMINAL INPUT/OUTPUT AREA (TIOA)

The statement

```
01 DFHTIOA COPY DFHTIOA.
```

copies the symbolic storage definition for the CICS/VS control section of the TIOA and must be present in all programs that use terminal input records or that provide output records to a terminal. The following is an example of the coding required to define the record(s) in the TIOA:

```
01 DFHTIOA COPY DFHTIOA.
   02 TRANSID PICTURE XXXX.
   02 TIOAMSG PICTURE X(20).
    .
    .
    .
```

The user must establish addressability for the TIOA in the Procedure Division by coding in the appropriate place either the statement

```
MOVE TCTTEDA TO TIOABAR.
```

or the statement

```
MOVE TCASCSA TO TIOABAR.
```

The former statement is used to establish addressability to a TIOA acquired dynamically by CICS/VS for data entered from a terminal. The latter statement is used to establish addressability for a new TIOA acquired dynamically through use of a DFHSC TYPE=GETMAIN macro instruction and should be coded on the line immediately following the last operand of that macro instruction.

FILE INPUT/OUTPUT AREA (FIOA)

The statement

```
01 DFHFIOA COPY DFHFIOA.
```

copies the symbolic storage definition for the CICS/VS control section of the FIOA and must be present in all programs requesting a read of an unblocked record without updating or segmenting, or a read of blocked records without deblocking. If desired, the user can identify the area returned in response to a file request as a FIOA, rather than a FWA or VSWA, by testing FIOAM. If data is retrieved using the Indexed Sequential Access Method (ISAM) under CICS/OS/VS, a 16-byte filler must be defined prior to the user's data definition. The following is an example of the coding required to define the record(s) in the FIOA:

```
01 DFHFIOA COPY DFHFIOA.
   02 FILLER PICTURE X(16).          NOTE OS/VS ISAM FILLER.
   02 KEYF PICTURE X(6).
   02 NAME PICTURE X(20).
   02 FIOAREC PICTURE X(74).
   .
   .
   .
```

The user must code the statement

```
MOVE TCAFCAA TO FIOABAR.
```

prior to any reference to the FIOA following a DFHFC macro instruction in the Procedure Division to establish addressability for the FIOA.

To identify the area returned as a FIOA, the following instruction can be used:

```
IF FIOAM, GO TO GOT-A-FIOA.
```

FILE WORK AREA (FWA)

The statement

```
01 DFHFWADS COPY DFHFWADS.
```

copies the symbolic storage definition for the CICS/VS control section of the FWA and must be present in all programs performing file operation with the exception of a "read without update" from an unblocked, unsegmented data set. If desired, the user can identify the area returned in response to a file request as a FWA, rather than FIOA or VSWA, by testing FWAM. The following is an example of the coding required to define the record(s) in the FWA:

```
01 DFHFWADS COPY DFHFWADS.
   02 KEYF PICTURE X(6).
   02 NAME PICTURE X(20).
   02 FWAREC PICTURE X(24).
   .
   .
   .
```

The user must code the statement

```
MOVE TCAFCAA TO FWACBAR.
```

prior to any reference to the FWA following a DFHFC macro instruction in the Procedure Division to establish addressability for the FWA.

To identify the area returned as a FWA, the following instruction can be used:

```
IF FWAM, GO TO GOT-A-FWA.
```

VSAM WORK AREA (VSWA)

The statement

```
01 DFHVSWA COPY DFHVSWA.
```

copies the symbolic storage definition for the CICS/VS control section of the VSAM work area and must be present in all programs using locate mode I/O. (See "File Services" in Chapter 6.) If desired, the user can identify the area returned in response to a file request as a VSWA, rather than a FIOA or FWA, by testing VSWAM. The user must code the statement

```
MOVE TCAFCAA TO VSWABAR.
```

prior to any reference to the VSWA acquired by CICS/VS in response to a DFHFC macro instruction utilizing locate mode I/O.

To identify the area returned as a VSWA, the following instruction can be used:

```
IF VSWAM, GO TO GOT-A-VSWA.
```

TRANSIENT DATA INPUT AREA (TDIA)

The statement

```
01 DFHTDIA COPY DFHTDIA.
```

copies the symbolic storage definition for the CICS/VS control section of the intrapartition TDIA and must be present in all programs requiring a message area for transient data obtained by issuing a DFHTD TYPE=GET macro instruction that references an intrapartition destination. (See "Acquire Queued Data (GET)" under "Transient Data Services" in Chapter 6.) The following is an example of the coding required to define the record(s) in the TDIA:

```
01 DFHTDIA COPY DFHTDIA.
   02 MESSAGE PICTURE X(25)
```

The user must code the statement

```
MOVE TCATDAA TO TDIABAR.
```

prior to any reference to the TDIA following a DFHTD macro instruction in the Procedure Division to establish addressability for the TDIA.

TRANSIENT DATA OUTPUT AREA (TDOA)

The statement

```
01 DFHTDOA COPY DFHTDOA.
```

copies the symbolic storage definition for the CICS/VS control section of the intrapartition TDOA and should be present in all programs issuing a DFHTC TYPE=PUT macro instructicn to provide transient data as output. (See "Dispose of Data (PUT)" under "Transient Data Services.") The following is an example of the coding required tc define the record(s) in the TDOA:

```
01 DFHTDOA COPY DFHTDOA.
   02 MESSAGE PICTURE X(20).
```

The user must code the statement

```
MOVE TCASCSA TO TDOABAR.
```

prior to any reference to the TDOA following a DFHSC macro instruction in the Procedure Division to establish addressability for the TDOA.

TEMPORARY STORAGE INPUT/OUTPUT AREA (TSIOA)

The statement

```
01 DFHTSIOA COPY DFHTSIOA.
```

copies the symbolic storage definition for the CICS/VS control section of the TSIOA and should be present in all programs using temporary storage. The following is an example of the coding required to define the record(s) in the TSIOA:

```
01 DFHTSIOA COPY DFHTSIOA.
   02 DATA PICTURE X(10).
```

To establish addressability fcr the TSIOA, the user must code the statements

```
MOVE TCATSDA TO TSICABAR.
SUBTRACT 8 FROM TSIOABAR.
```

if the request is a GET or GETQ frcm temporary storage and the TSDADDR operand is not specified. The subtraction of eight ensures that TSIOABAR points to the storage accounting field (that is, to the beginning) of the storage area acquired by CICS/VS. He must code the statement

```
MOVE TCASCSA TO TSIOABAR.
```

if he has dynamically acquired an I/O area. In the case of a PUT or PUTQ, the symbolic address of the data is located at TSIOAVRL. Either statement must appear in the appropriate place in the Procedure Division of the ANS COBOL program.

STORAGE ACCOUNTING AREA (SAA)

The statement

```
01 DFHSAADS COPY DFHSAADS.
```

copies the symbolic storage definition for the SAA. This storage definition should precede the definition of user storage acquired through the DFHSC TYPE=GETMAIN,CLASS=USER macro instruction. The following is an example of the coding required to define the record(s) in the SAA:

```
01 DFHSAADS COPY DFHSAADS.
   02 NAME PICTURE X(20).
   02 SAAREC PICTURE X(10).
   .
   .
   .
```

The user must code the statement

```
MOVE TCASCSA TO SAACBAR.
```

prior to any reference to the SAA following a DFHSC macro instruction in the Procedure Division to establish addressability for the SAA.

JOURNAL CONTROL AREA (JCA)

The statement

```
01 DFHJCADS COPY DFHJCADS.
```

copies the symbolic storage definition for the CICS/VS control section of the journal control area (JCA) and must be present in all programs requesting journal services. (See "Journal Services" in Chapter 6.)

A JCA is acquired dynamically by means of a DFHJC TYPE=GETJCA macro instruction. Addressability to the JCA is provided automatically through the macro expansion, which loads the address of the area into JCABAR.

ADDITIONAL GUIDELINES

If the ANS COBOL programmer wishes to access a series of chained storage areas (each area contains a pointer to the next area in the chain), he must establish addressability to each new storage area in the chain by inserting a paragraph name immediately following any MOVE statement that establishes addressability. For example:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   .
   .
   .
   02 USERPTR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   .
   .
   .
01 DFHTCADS COPY DFHTCADS.
   02 TWAFIELD PICTURE X(4).
   .
   .
   .
01 USERAREA.
   02 FIELD PICTURE X(4).
   02 NEXTAREA PICTURE S9(8) USAGE IS COMPUTATIONAL.
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
   MOVE NEXTAREA TO USERPTR.
ANYNAME.
```

```
MOVE FIELD TO TWAFIELD.
```

In this example, storage areas mapped or defined by USERAREA are chained. The first MOVE instruction establishes addressability to the next area in the chain. The second MOVE instruction moves data from the newly addressed area, but only because the paragraph name precedes the second MOVE instruction; in the absence of the paragraph name, data is moved from the previously addressed area rather than from the new area. Note that a paragraph name is not needed if addressability to an area is obtained through a field in some other area (for example, the TCA).

If the object of an OCCURS DEPENDING ON clause is defined in the linkage section, special consideration is required to ensure that the correct value is used at all times. In the following example, FIELD-COUNTER is defined in the linkage section. The MOVE FIELD-COUNTER TO FIELD-COUNTER statement is needed to ensure that unpredictable results do not occur when referencing DATA.

```
LINKAGE SECTION.
01 DFHFWADS COPY DFSFWADS.
   .
   .
   .
   02 FIELD-COUNTER PIC 9(4) USAGE IS COMPUTATIONAL.
   02 FIELDS PIC X(5) OCCURS 1 TO 5 TIMES
      DEPENDING ON FIELD-COUNTER.
   02 DATA PIC X(20).
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
   DFHFC TYPE=GET, etc.
   MOVE TCAFCAA TO FWACBAR.
   MOVE FIELD-COUNTER TO FIELD-COUNTER.
   MOVE DATA TO TWA-FIELD.
```

The MOVE statement referring to FIELD-COUNTER causes ANS COBOL to reestablish the value it uses to compute the current number of occurrences of FIELDS and ensures that it can correctly determine the displacement of DATA.

If an area greater than 4095 bytes is defined in the linkage section, special considerations arise: An additional 02-level statement under DFHBLLDS and an ADD statement following the MOVE statement to establish addressability to the area are required for each additional 4096 bytes. For example, if a file work area (FWA) exceeds 4095 bytes, the following code can be used.

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   .
   .
   .
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   02 FWABR1 PICTURE S9(8) USAGE IS COMPUTATIONAL.
   .
   .
   .
01 DFHFWADS COPY DFHFWADS.
   02 FIELD1 PICTURE X(4000).
   02 FIELD2 PICTURE X(1000).
```

```
           02 FIELD3 PICTURE X(400).
              .
              .
              .
       PROCEDURE DIVISION.
              .
              .
              .
           DFHFC TYPE=GET,                                          *
              .
              .
              .
           MOVE TCAFCAA TO FWACBAR.
           ADD 4096 TO FWACBAR GIVING FWABR1.
```

If an application program is to be compiled for execution under CICS/OS/VS using the full ANS COBOL V4 Compiler (5734-CB2), the OS/VS COBOL Compiler (5740-CB1), or the DOS/VS COBOL Compiler (5746-CB1) with the optimization (OPT) feature, a special compiler control statement must be inserted at appropriate places within the program to ensure addressability to a particular area defined in the Linkage section. This control statement has the form:

```
SERVICE RELOAD fieldname.
```

where fieldname is the symbolic name of a specific storage area, and is also defined in an 01-level statement in the linkage section. The first four statements of the Procedure Division must be

```
SERVICE RELOAD DFHBLLDS.
SERVICE RELOAD DFHCSADS.
MOVE CSAOPFLA TO CSAOPBAR.
SERVICE RELOAD CSAOPFI.
```

Statements such as

```
MOVE TCAFCAAA TO TCTTEAR.
SERVICE RELOAD DFHTCTTE.
```

or

```
SUBTRACT 8 FROM TCASCSA GIVING TSIOABAR.
SERVICE RELOAD DFHTSICA.
```

can be used to establish addressability for a particular storage area. (Note that the SERVICE RELOAD statement must be used following each statement which modifies addressability to an area defined in the linkage section, that is, whenever an address is moved to a field named in an 02-level statement under 01 DFHBLLDS or the address in the 02-level statement is changed in any way.)

To establish addressability to the TCA, the following statements must be coded:

```
MOVE CSACDTA TO TCACBAR.
SERVICE RELOAD DFHTCA.
```

Note that the RELOAD statement specifies DFHTCA, not DFHTCADS.

Certain ANS COBOL features cannot be used in an application program to be run under CICS/VS. Generally, these features are replaced by CICS/VS services. They are identified under "Restrictions" in the first chapter of this manual.

EXAMPLE OF CICS/VS ANS COBOL APPLICATION PROGRAM

Figure 4-1 is an ANS COBOL program written to run under CICS/VS. The program asks a question of the terminal operator, receives a reply, dynamically acquires some storage, and sends the operator's message back to the terminal. In effect, an echo test is performed. (The line numbers are for discussion purposes only; they are not part of the program.)

```
01      IDENTIFICATION DIVISION.
02      PROGRAM-ID.
03         'CBLSPRB'.
04      ENVIRONMENT DIVISION.
05      DATA DIVISION.
06      LINKAGE SECTION.
07      01 DFHBLLDS COPY DFHBLLDS.
08         02 TCTTEAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
09         02 TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
10      01 DFHCSADS COPY DFHCSADS.
11      01 DFHTCADS COPY DFHTCADS.
12         02 SAVE-LENGTH PICTURE S9(8) USAGE IS COMPUTATIONAL.
13         02 SAVE-MESSAGE PICTURE X(32).
14      01 DFHTCTTE COPY DFHTCTTE.
15      01 DFHTIOA COPY DFHTIOA.
16         02 TIOAMSG PICTURE X(32).
17      PROCEDURE DIVISION.
18         MOVE CSACDTA TO TCACBAR.
19         MOVE CSAOPFLA TO CSAOPBAR.
20         MOVE TCAFCAAA TO TCTTEAR.
21         MOVE TCTTEDA TO TIOABAR.
22         MOVE 'IS THIS A COBOL OR A PL/I PROGRAM' TO TIOAMSG.
23         MOVE 33 TO TIOATDL.
24      DFHTC TYPE=(WRITE,REAL,WAIT)
25         MOVE TCTTEDA TO TIOABAR.
26         MOVE TIOATDL TO SAVE-LENGTH.
27         MOVE TIOAMSG TO SAVE-MESSAGE.
28      DFHSC TYPE=GETMAIN,                                          *
29            NUMBYTE=32,                                           *
30            INITIMG=40,                                           *
31            CLASS=TERMINAL
32         MOVE TCASCSA TO TICABAR.
33         MOVE TIOABAR TO TCTTEDA.
34         MOVE SAVE-MESSAGE TO TIOAMSG.
35         MOVE SAVE-LENGTH TO TIOATDL.
36      DFHTC TYPE=WRITE
37      DFHPC TYPE=RETURN
38         GOBACK.
```

Figure 4-1. Example of CICS/VS ANS COBOL Application Program

A discussion of the significance of each of the lines of Figure 4-1 follows.

Line Number	Description
01-05	Required for ANS COBOL.
06	Start of linkage section.
07	Copies symbolic storage definition for BLL; contains addresses of CICS/VS storage areas.
08-09	Adds addresses for TCTTE and TIOA (required for statements 14 and 15).
10	Copies symbolic storage definition for CSA.
11	Copies symbolic storage definitions for TCA

	and CSA optional features list.
12-13	Defines save areas in TWA to ensure quasi-reentrance (SAVE-LENGTH and SAVE-MESSAGE are used to save operator's reply).
14	Copies symbolic storage definition for TCTTE.
15	Copies symbolic storage definition for TIOA.
16	Defines message area in TIOA.
17	Required for ANS COBOL (start of Procedure Division).
18-21	Establishes addressability for TCA, CSA optional features list, TCTTE, and TIOA (CICS/VS establishes addressability for BLL and CSA).
22	Moves message to output area of TIOA.
23	Moves length of message to data length field of TIOA.
24	CICS/VS macro instruction that writes message to terminal, waits for operator's reply, and reads operator's reply.
25	Establishes addressability for new TIOA using address in TCTTE.
26	Saves length of message in TWA.
27	Saves message in TWA.
28-31	CICS/VS macro instruction that requests 32 bytes of terminal storage initialized to blanks (terminal storage is chained to terminal control table).
32	Establishes addressability for new TIOA (address of newly acquired storage area is in TCASCSA field of the TCA).
33	Places address of new TIOA in terminal control table.
33	Moves message to output area (TIOA).
35	Moves length of message to cutput area (TIOA).
36	CICS/VS macro instruction that writes message to terminal.
37	CICS/VS macro instruction that returns control to CICS/VS.
38	ANS COBOL statement that marks the end of the program.

CHAPTER 5. COPYING STORAGE DEFINITIONS - PL/I

The PL/I programmer must define storage for the CICS/VS control areas and any other storage areas required for the processing of his program. He accomplishes this by using a statement of the form

```
%INCLUDE (name);
    or
%INCLUDE name;
```

to (1) copy the appropriate symbolic storage definition into his program at the place where the %INCLUDE statement appears, and (2) specify the name of the storage area being defined.

The source code provided by CICS/VS in response to %INCLUDE statements is in the form of based structures. These structures describe the attributes of the storage areas and include pointer variables that provide the addresses of the actual locations in storage that the structures describe.

All programs must contain statements to copy the symbolic storage definitions for the common system area (CSA) and task control area (TCA). CICS/VS macro expansions resulting from macro instructions that the application programmer uses refer to fields within these areas, so their locations must be identified. Whether additional storage definitions must be copied depends on the processing requirements (storage areas and macro instructions used) of the application program.

A PL/I program to be run under CICS/VS must contain the REENTRANT option in the initial PROCEDURE statement to satisfy the CICS/VS requirement that code be quasi-reentrant. See "Restrictions" in the first chapter of this manual for a listing of PL/I features that cannot be used.

STATIC STORAGE DEFINITION

During CICS/VS initialization, the common system area (CSA) is statically allocated as part of the CICS/VS nucleus. For each terminal with which communication is to occur, the terminal control table terminal entry (TCTTE) is included in the statically allocated terminal control table (TCT). The PL/I programmer must provide symbolic storage definitions for the CSA and TCTTE (if needed) as follows.

COMMON SYSTEM AREA (CSA)

The statement

```
%INCLUDE (DFHCSADS);
```

copies the based structures that symbolically define the CSA and the CSA optional features list. Addressability for both areas is included.

If CICS/VS was generated to support a common work area (CWA) within the CSA, the PL/I programmer can provide, immediately following the %INCLUDE (DFHCSADS) macro instruction, coding such as the following:

```
DECLARE 1 DFHCSAWK BASED (CSACBAR),
    2 CSAFILL CHAR(512),
    2 USERLBL1 attributes,
    .
    .
    .
    2 USERLBLn attributes;
```

TERMINAL CONTROL TABLE TERMINAL ENTRY (TCTTE)

The statement

```
%INCLUDE (DFHTCTTE);
```

copies the based structure that symbolically defines the TCTTE and must be present in all programs requesting communication with a terminal. Addressability for the TCTTE is included.

DYNAMIC STORAGE DEFINITION

During initiation and execution of a transaction (task), the task control area (TCA), terminal input/output area (TIOA), and other storage areas required by the transaction are dynamically allocated by CICS/VS storage management, upon request from either the application program or a CICS/VS management function. The PL/I programmer must provide symbolic definitions for these storage areas as follows.

TASK CONTROL AREA (TCA)

The statement

```
%INCLUDE (DFHTCADS);
```

copies the based structure that symbolically defines the TCA and establishes addressability.

The latter part of the based structure consists of a DECLARE statement that is not terminated by a semicolon. The user must complete the declaration of the TCA structure by supplying an ending (for example, a semicolon) or, if a transaction work area (TWA) is desired, by supplying further declaration. The following is an example of the coding required:

```
%INCLUDE (DFHTCADS);
   2 TWA CHAR(40);
   .
   .
   .
```

TERMINAL INPUT/OUTPUT AREA (TIOA)

The statement

```
%INCLUDE (DFHTIOA);
```

copies the based structure that symbolically defines the CICS/VS control section of the TIOA and establishes addressability. This statement must be present in all programs that use terminal input records or that write output records to a terminal. The application programmer must complete the declaration of the TIOA structure by supplying an ending (for example, a semicolon) or by supplying further declaration of the

input/output area. He can then request an action that requires a TIOA. For example, he can issue a DFHSC TYPE=GETMAIN macro instruction requesting CICS/VS storage contrcl to obtain dynamic storage for a TIOA for his program. The following is an example of the coding required:

```
%INCLUDE (DFHTIOA);
   2 NAME CHAR(20),
   2 STREET CHAR(20);
   .
   .
   .
DFHSC TYPE=GETMAIN,                                              *
      NUMBYTE=XX,                                                *
      CLASS=TERMINAL
TIOABAR=TCASCSA; /* TCASCSA FIELD OF TCA CONTAINS ADDRESS
                    OF NEWLY ACQUIRED STORAGE */
   .
   .
   .
```

For additional information about GETMAIN, see "Obtain and Initialize Main Storage (GETMAIN)" under "Storage Services" in Chapter 6.

FILE INPUT/OUTPUT AREA (FIOA)

The statement

```
%INCLUDE (DFHFIOA);
```

copies the based structure that symbolically defines the CICS/VS control section of the FIOA and must be present in all programs requesting a read of an unblocked record without updating or segmenting, or a read of blocked records without deblocking. If desired, the user can identify the area returned in response to a file request as a FIOA, rather than a FWA or VSWA, by testing FIOAIND for a bit value of 01. The application programmer must complete declaration of the FIOA. He must establish addressability for the FIOA using the statement

```
FIOABAR=TCAFCAA;
```

following the DFHFC macro instruction. If data is retrieved using the Indexed Sequential Access Method (ISAM) under CICS/OS/VS, a 16-byte filler must be defined prior to the user's data definition. The following example illustrates the coding required, as well as the optional coding for FIOA identification:

```
%INCLUDE (DFHFIOA);
   2 FILL CHAR(16),              /*OS/VS ISAM FILLER*/
   2 NAME CHAR(20),
   2 ADDR CHAR(20);
   .
   .
   .
FIOABAR=TCAFCAA;
IF FIOAIND='01'B THEN GO TO GOT_A_FIOA;
   .
   .
   .
```

FILE WORK AREA (FWA)

The statement

```
%INCLUDE (DFHFWADS);
```

copies the based structure that symbolically defines the CICS/VS control section of the FWA. This statement should precede a user-declared file record area when reading or updating an existing blocked or segmented record, when adding a new record to a data set, or when retrieving records using the browse technique. If desired, the user can identify the area returned in response to a file request as a FWA, rather than a FIOA or VSWA, by testing FWAIND for a bit value of 11. The user must complete declaration of the FWA. He must establish addressability for the FWA using the statement

```
FWACBAR=TCAFCAA;
```

following a DFHFC macro instruction. The following example illustrates the coding required, as well as the optional test for FWA identification:

```
%INCLUDE (DFHFWADS);
   2 NAME CHAR(20),
   2 ADDR CHAR (20);
   .
   .
   .
FWACBAR=TCAFCAA;
IF FWAIND='11'B THEN GO TO GOT_A_FWA;
   .
   .
   .
```

VSAM WORK AREA (VSWA)

The statement

```
%INCLUDE (DFHVSWA);
```

copies the based structure that symbolically defines the CICS/VS control section of the VSAM work area and must be present in all programs using locate mode I/O. (See "File Services" in Chapter 6.) If desired, the user can identify the area returned in response to a file request as a VSWA, rather than a FIOA or FWA, by testing VSWAID for a bit value of 00000000. The user must establish addressability for the VSWA using the statement

```
VSWABAR=TCAFCAA;
```

following the DFHFC macro instruction utilizing locate mode I/O which causes CICS/VS to acquire the VSWA.

To identify the area returned as a VSWA, the following instruction can be used:

```
IF VSWAID='00000000'B THEN GO TO GOT_A_VSWA;
```

TRANSIENT DATA INPUT AREA (TDIA)

The statement

```
%INCLUDE (DFHTDIA);
```

copies the based structure that symbolically defines the CICS/VS control section of the intrapartition TDIA and must be present in all programs requiring a message area for transient data obtained by issuing a DFHTD TYPE=GET macro instruction that references an intrapartition destination. (See "Acquire Queued Data (GET)" under "Transient Data Services" in Chapter 6.) The user must complete declaration of the TDIA. He must establish addressability for the TDIA using the statement

```
TDIABAR=TCATDAA;
```

following a DFHTD macro instruction. The following is an example of the coding required:

```
%INCLUDE (DFHTDIA);
   2 MSG CHAR(40);
   .
   .
   .
TDIABAR=TCATDAA;
   .
   .
   .
```

TRANSIENT DATA OUTPUT AREA (TDOA)

The statement

```
%INCLUDE (DFHTDOA);
```

copies the based structure that symbolically defines the CICS/VS control section of the intrapartition TDOA and should be present in all programs issuing a DFHTC TYPE=PUT macro instruction to provide transient data as output. (See "Dispose of Data (PUT)" under "Transient Data Services".) The user must complete declaration of the TDOA. He must establish addressability for the TDOA using the statement

```
TDOABAR=TCASCSA;
```

following a DFHSC macro instruction. The following is an example of the coding required:

```
%INCLUDE (DFHTDOA);
   2 TIME CHAR(2),
   2 DATA CHAR(3),
   2 INTERM CHAR(4),
   2 OUTTERM CHAR(4);
   .
   .
   .
DFHSC TYPE=GETMAIN,                                          *
      NUMBYTE=XX,                                            *
      CLASS=USER
TDOABAR=TCASCSA;
   .
   .
   .
```

TEMPORARY STORAGE INPUT/OUTPUT AREA (TSIOA)

The statement

```
%INCLUDE (DFHTSIOA);
```

copies the based structure that symbolically defines the CICS/VS control section of the TSIOA and must be present in all programs using temporary storage. The application programmer must complete declaration for the TSIOA. He must establish addressability for the TSIOA using coding such as:

```
DCL TSIOABAA FIXED BIN(30) BASED(TSIOABAB);
    TSIOABAR=TCATSDA;
    TSIOABAB=ADDR(TSICABAR);
    TSIOABAA=TSIOABAA - 8;
```

if the request is a GET or GETQ frcm temporary storage and the TSDADDR operand is not specified. The subtraction of eight ensures that TSIOABAA points to the storage accounting field (that is, to the beginning) of the storage area acquired by CICS/VS. He must code the statement

```
TSIOABAR=TCASCSA;
```

if he has dynamically acquired the I/O area. In the case of a PUT or PUTQ, the symbolic address of the data is located at TSIOAVRL.

STORAGE ACCOUNTING AREA (SAA)

The statement

```
%INCLUDE (DFHSAADS);
```

copies the based structure that symbolically defines the SAA and must be present in all programs requesting storage through use of the DFHSC TYPE=GETMAIN,CLASS=USER macro instruction. This statement must precede the definition of user storage. The application programmer must complete declaration for the SAA and establish addressability for the SAA using the statement

```
SAACBAR=TCASCSA;
```

The following example illustrates the coding required:

```
%INCLUDE (DFHSAADS);
   2 MSG CHAR(40);
   .
   .
   .
DFHSC TYPE=GETMAIN,                                          *
      NUMBYTE=60,                                            *
      CLASS=USER
SAACBAR=TCASCSA;
   .
   .
```

JOURNAL CONTROL AREA (JCA)

The statement

```
%INCLUDE (DFHJCADS);
```

copies the based structure that symbolically defines the CICS/VS control section of the journal control area (JCA) and must be present in all programs requesting journal services. (See "Journal Services" in Chapter 6.)

A JCA is acquired dynamically by means of a DFHJC TYPE=GETJCA macro instruction. Addressability to the JCA is provided automatically through the macro expansion, which loads the address of the area into JCABAR.

EXAMPLE OF CICS/VS PL/I APPLICATION PROGRAM

Figure 5-1 is a PL/I program written to run under CICS/VS. The program asks a question of the terminal operator, receives a reply, dynamically acquires some storage, and sends the operator's message back to the terminal. In effect, an echo test is performed. (The line numbers are for discussion purposes only; they are not part of the program.)

```
01     PL1PROG: PROCEDURE OPTIONS (MAIN,REENTRANT);
02     %INCLUDE (DFHCSADS);
03     %INCLUDE (DFHTCADS);
04        2 SAVE_LENGTH BINARY FIXED (15),
05        2 SAVE_MSG CHAR (32);
06     %INCLUDE (DFHTCTTE);
07     %INCLUDE (DFHTIOA);
08        2 TIOAMSG CHAR(32);
09     TIOAMSG='IS THIS A COBOL OR A PL/I PROGRAM';
10     TIOATDL=33;
11     DFHTC TYPE=(WRITE,REAL,WAIT)
12     TIOABAR=TCTTEDA;
13     SAVE_LENGTH=TIOATDL;
14     SAVE_MSG=TIOAMSG;
15     DFHSC TYPE=GETMAIN,                                          *
16           NUMBYTE=32,                                            *
17           INITIMG=40,                                            *
18           CLASS=TERMINAL
19     TIOABAR=TCASCSA;
20     TCTTEDA=TIOABAR;
21     TIOAMSG=SAVE_MSG;
22     TIOATDL=SAVE_LENGTH;
23     DFHTC TYPE=WRITE
24     DFHPC TYPE=RETURN
25     END;
```

Figure 5-1. Example of CICS/VS PL/I Application Program

A discussion of the significance of each of the lines of Figure 5-1 follows.

Line Number	Description
01	Required for PL/I. REENTRANT option specified to meet requirement of CICS/VS that code be quasi-reentrant.
02	Retrieves symbolic storage definitions for CSA and CSA optional features list and establishes addressability.
03	Retrieves symbolic storage definition for TCA and establishes addressability.
04-05	Defines the TWA and terminates the DECLARE statement. SAVE_MSG and SAVE_LENGTH are used to preserve the operator's reply.
06	Retrieves symbolic storage definition for TCTTE and establishes addressability.
07	Retrieves symbolic storage definition for TIOA

Line Number	Description
	and establishes addressability.
08	Describes I/O area for terminal message and terminates the DECLARE statement.
09	Places message to be sent to operator in the TIOA.
10	Places the message length in the terminal data length field of the TIOA.
11	Writes the message to the terminal, waits for the operator's reply, and reads the operator's reply.
12	Reestablishes addressability for the TIOA using address in the TCTTE.
13-14	Saves the operator's message and message length in the TCA.
15-18	CICS/VS macro instruction that requests 32 bytes of terminal storage initialized to blanks (terminal storage is chained to terminal control table).
19	Establishes addressability for the new TIOA (the address of the newly acquired storage is in the TCASCSA field of the TCA).
20	Places address of new TIOA in terminal control table.
21-22	Moves message and length of message to output area (TIOA).
23	CICS/VS macro instruction that sends operator's message back to the terminal.
24	CICS/VS macro instruction that returns control to CICS/VS.
25	PL/I statement that marks the end of the procedure.

CHAPTER 6. SERVICE INVOCATION

The CICS/VS system management component provides the following supervisory and data management services:

- Terminal services - Terminal Management/Terminal Control program
- File services - File Management/File Control program
- Transient data services - Transient Data Management/Transient Data Control program
- Temporary storage services - Temporary Storage Management/Temporary Storage Control program
- Storage services - Storage Management/Storage Control program
- Program services - Program Management/Program Control program
- Time services - Time Management/Interval Control program
- Task services - Task Management/Task Control program
- Journal services - Journal Management/Journal Control program
- Restart/recovery services - Sync Point Management/Sync Point program

Each program performs the following basic steps:

1. Analyzes a specific service request issued by an application program or another CICS/VS program
2. Performs the requested service by communicating with the operating system, as necessary, through macro instructions
3. Retains the status of each service request until the service is provided
4. Maintains statistical information that can be used to evalulate system performance

This chapter, building on the introduction of Chapter 1, explains these services and how the application programmer uses macro instructions to request them. When preparing programs for CICS/VS, however, the programmer should refer to the discussions of specific macro instructions in Chapter 7.

TERMINAL SERVICES

Terminal management provides for communication between terminals and user-written application programs through terminal control. Terminal control is responsible for the polling and addressing of terminals, code translation, transaction initiation, task and line synchronization, and the line control necessary to read from or write to a terminal. The user-written application program is thus relieved, as much as possible, from having to control the physical terminal environment.

Requests for terminal services are communicated directly to terminal control through CICS/VS macro instructions. Individual application

programs thus interface with a terminal both logically and symbolically. Before issuing a DFHTC macro instruction, the application programmer must make sure that TCAFCAAA contains a terminal control table terminal entry (TCTTE).

Terminal control operates as a system-provided task, under control of its own TCA. It contends for system resources with user-provided tasks in the system, but it is the highest-priority task in CICS/VS. Terminal control is always the first task to be dispatched by CICS/VS; it scans the service request indicators in the TCT and performs requested services.

In providing for communication between user terminals and user-written application programs, CICS/VS uses standard access methods available with the host operating system (DOS/VS or OS/VS). The Basic Telecommunications Access Method (BTAM) is used by CICS/VS for terminal management of most start-stop and BSC terminals. As an option for OS/VS, the Telecommunications Access Method (TCAM) can be specified. The Sequential Access Method (SAM) is used where key-driven terminals are simulated by sequential devices such as card readers and line printers. The Virtual Telecommunications Access Method (VTAM) is used for terminal management of system network architecture (SNA) terminal systems.

Note: The last data character in each logical input record from a sequential device must be followed immediately by a character defined as an end-of-block (EOB) character by the system programmer. For sequential devices, the last entry in the input stream should be CSSF (control system sign-off) GOODNIGHT or CSOT RECV to provide a logical close. If end of data (EOD) is encountered, all subsequent reads are treated as errors and only writes are processed. There may be multiple logical records on a single physical record (multiple EOD characters on a single card). For users of CICS/OS/VS having blocked SYSIN or SYSOUT, overriding DD statements must be provided to specify unblocked for CICS/VS data sets used to simulate terminals.

The terminal management macro instruction (DFHTC) is used to request any of a wide variety of services. Among them are some services of interest to users at terminals of most, if not all, terminal types supported by CICS/VS. These include:

- Write data to a terminal
- Read data from a terminal
- Synchronize terminal input/output for a transaction
- Converse with a terminal
- Read or write multiple records to a card reader, disk data set, magnetic tape unit, or line printer defined by the system programmer as a card-reader-in-line-printer-out (CRLP) terminal.

For additional information concerning the last of these services, see "Sequential Terminal Support" in Chapter 8.

Other services performed in response to DFHTC macro instructions are applicable to specific terminal types. Since many types of terminals are supported by CICS/VS, many special services are provided. (For a list of terminals supported by CICS/VS, see the CICS/VS General Information Manual.) The list below is representative of the terminal-oriented input/output services available:

- Read the entire contents of a buffer (3270 Information Display System)

- Read a message containing both uppercase and lowercase data (3270 Information Display System)
- Print out the contents of an information display buffer on a printer (3270 Information Display System)
- Transmit a message to a common buffer (2980 General Banking System)
- Read or write data in transparent mode, that is, without translation (System/7, System/370, System/3, 2770 Data Communication System, 2780 Data Transmission Terminal, 3740 Data Entry System, 3780 Data Communication Terminal).
- Use the Attention key to interrupt a write operation or signal a read attention request (2741 Communications Terminal)

2260-compatibility support by means of the 3270 Information Display System allows the user to run 2260-based transactions developed for preceding versions of CICS from a 3270. To make this support available, an installation need only request that 2260 compatibility be included during CICS/VS system generation. This generates the code necessary to convert 2260 data streams from user-written application programs to the appropriate 3270 data stream format, or 3270 to 2260. A task using this capability is operating in compatibility mode.

The general form of the terminal management macro instruction (DFHTC) resembles that of other CICS/VS macro instructions. One or more keyword parameters are specified, separated by commas. Whereas most CICS/VS macro instructions use only one entry following the keyword TYPE, the DFHTC macro instruction may contain several. For example, the

```
DFHTC TYPE=(WRITE,READ)
```

macro instruction causes a write to the terminal, a wait for that write to be completed (an implied wait), and a read from the terminal to which data has just been written.

As another example, the

```
DFHTC TYPE=(WRITE,ERASE,READ,WAIT)
```

macro instruction causes an erase and then a write to a terminal, followed by an implied wait, followed by a read and a requested wait. This latter wait ensures that the read is complete before control is returned to the application program.

The order in which operand entries are specified is not significant. That is, the

```
DFHTC TYPE=(ERASE,WRITE,WAIT,READ)
```

macro instruction has the same effect as the example above. Each entry, independent of its position, affects the setting of an associated bit in the terminal control table terminal entry (TCTTE) for the terminal from which the macro instruction is entered. Therefore, the order in which entries are specified has no effect on their meaning. The application programmer cannot request a read and then a write by means of one DFHTC macro instruction, but he can easily set up two DFHTC macro instructions to do so.

As in other CICS/VS macro instruction operands, if only one entry is given in the TYPE operand, no parentheses are necessary. For example, the

DFHTC TYPE=READ

macro instruction is used to request that data be read from a terminal.

The application programmer must determine the combination of keywords that follows TYPE=, depending on the terminal (and sometimes, access method) used and the operation(s) to be performed. Additional operands may be required or desired, again depending on the terminal and access method used. Some common input/output requests are discussed below. (For full details of DFHTC, see Chapter 7; for specific terminals, see Chapter 11; for further VTAM considerations, see the CICS/VS Advanced Communication Guide, SH20-9049.)

Before terminal services can be requested by means of the DFHTC macro instruction, the application programmer must provide instructions that do the following:

1. Symbolically define the TCTTE and TIOA by copying the appropriate storage definitions (DFHTCTTE and DFHTIOA) provided by CICS/VS. (It is assumed that the storage definitions for the CSA and TCA have already been copied, as described in Chapters 2 through 5.)

2. Establish addressability for the TCTTE and TIOA by specifying a symbolic base address for the TCTTE and TIOA, respectively. The Assembler-language or American National Standard (ANS) COBOL application programmer must obtain the base address of the TCTTE from TCAFCAAA and place it in TCTTEAR; with PL/I, addressability for the TCTTE is established automatically. After addressability to the TCTTE has been established, the application programmer must obtain the base address of the TIOA from TCTTEDA and place it at TIOABAR. Any field in the TCTTE or TIOA can then be accessed by field name.

CICS/VS allows one or more TIOAs to be associated with a terminal at a given time. If a TIOA is obtained through the DFHSC TYPE=GETMAIN,CLASS=TERMINAL macro instruction, the address of the TIOA is automatically placed in TCASCSA. The application programmer must set up a base register for this TIOA and place the address at TCASCSA into the base register.

The length of the data to be read or written into a given TIOA is found in TIOATDL. On a read operation, this two-byte binary value is placed in TIOATDL by terminal control and represents the number of bytes actually read. On a write operation, the application programmer must place the number of bytes to be written in TIOATDL prior to issuing the DFHTC TYPE=WRITE macro instruction.

TIOAs and terminal control macro instructions are used by CICS/VS basic mapping support (BMS) in response to application-program requests for BMS services. However, for BMS operations that use mapping, the application programmer need not be concerned with providing the length of the data in TIOATDL prior to output; note that this does not apply to the DFHBMS TYPE=TEXTBLD macro instruction, because it does not involve mapping. After a BMS input operation, TIOATDL contains zeros.

Note: All TIOAs have a 12-byte prefix for storage accounting and terminal control and a one-byte terminating suffix. The value at TIOATDL does not include these 13 bytes.

WRITE DATA TO A TERMINAL (WRITE)

The application programmer can request that data be written to a terminal by issuing the

DFHTC TYPE=WRITE

macro instruction. Before issuing this macro instruction, the address of the TIOA from which data is to be written must be placed into TCTTEDA and the length of the data to be written into TIOATDL. (Of course, the CSA, TCA, and TCTTE must also be symbolically defined and, with Assembler language or ANS COBOL, addressability established for the TCTTE.)

When the write is completed by terminal control, the TIOA is released to a dynamic storage pool (unless SAVE is specified). Any subsequent reference to this TIOA by the application program is logically in error and produces unpredictable results.

A TIOA can be reused by the application program after a write if the request to write data to a terminal is made through the

DFHTC TYPE=(WRITE,SAVE,WAIT)

macro instruction. In this case the TIOA is not released by terminal control. The WAIT parameter is needed to ensure that the write of the TIOA is complete before the area is reused.

Note: To ensure a dump of the TIOA following a terminal control write, the application programmer must issue a SAVE and WAIT with the DFHTC TYPE=WRITE macro instruction that precedes the DFHDC macro instruction.

As pointed out earlier, the application programmer can specify a write followed by a read operation in a single request by issuing the

DFHTC TYPE=(WRITE,READ)

macro instruction. A typical use for this macro instruction occurs in a conversational environment in which the application program writes a question to the terminal, waits for a response, and then reads the response. Since the SAVE parameter is not specified, terminal control may reuse the TIOA from which data is written as a TIOA for the input data. However, a new TIOA is obtained for the read operation and its address placed in TCTTEDA when certain devices are involved or when certain conditions exist. For example:

- Local 3270 terminals.
- PSEUDOBIN is specified with READ,WRITE.
- The TIOA length for the WRITE instruction is less than that specified in the DFHTCT TYPE=TERMINAL,TIOAL=length specification (binary synchronous terminals) or in the DFHTCT TYPE=LINE, INAREAL=length specification (all other terminals) by the system programmer.
- Certain error conditions.
- A 3270 terminal is used in 2260 compatibility mode.

Thus the user should always reload TIOABAR from TCTTEDA following the (WRITE,READ) macro instruction.

Note: In the case of a terminal connected to the 7770 Audio Response Unit, a read request that does not include the WRITE parameter causes the "ready" message defined in the terminal control table by the system programmer to be written to the terminal before the read operation occurs.

If both a write and a read operation are specified in a single request by issuing the

```
DFHTC TYPE=(WRITE,REAL,SAVE)
```

macro instruction, the TIOA used for writing is saved; a new TIOA is then dynamically acquired by terminal control for the read. If the saved TIOA is reused later for either writing or reading, the application programmer must place the address of the TIOA into TCTTEDA prior to issuing the request to use the area.

The manner in which the address of a TIOA is "remembered" is a responsibility of the application programmer. Generally, the address should be in the TWA.

Upon completion of a (WRITE,REAL,SAVE), the application programmer must place the value contained at TCTTEDA into TIOABAR to establish addressability for the newly acquired TIOA.

Note: If a (WRITE,READ,SAVE) is issued, CICS/VS dynamically acquires a TIOA for the read. The size of that TIOA is determined by system programmer specifications for the terminal control table terminal entry for the terminal (rather than on the size of the TIOA used for the write).

If a write to a 3270 terminal (operating in 2260 compatibility mode) is specified by issuing the

```
DFHTC TYPE=(WRITE,ERASE)
```

macro instruction, the screen is erased and the cursor is returned to the upper left corner of the screen before writing occurs. If the ERASE parameter is omitted, writing begins wherever the cursor is located at the time the write is issued. To simply erase the screen, the application programmer might (1) place at TCTTEDA the address of a TIOA, (2) place at TIOATDL a data length of 0, and (3) issue a DFHTC TYPE=(WRITE,ERASE) macro instruction. If operating in 2260 compatibility mode, the TIOA should only contain a start symbol and the data length in TIOATDL should be set to 1 before issuing the DFHTC TYPE=(WRITE,ERASE).

Sometimes it is desirable that information displayed on the screen of a 3277 or 3275 be written out on a printer (provided as "hard copy") as well. If print request support is generated into CICS/VS, the application programmer can issue a

```
DFHTC TYPE=PRINT
```

macro instruction to cause the data currently displayed on a 3277 or 3275 to be printed out on an eligible 3284 or 3286 terminal. Such a terminal is eligible only if FEATURE=PRINT was specified by the system programmer in setting up the TCTTE. (For additional considerations, see "3270 Print Function" in Chapter 11 and the CICS/VS System Programmer's Reference Manual.)

If 2741 write break support is generated into CICS/OS/VS, the application programmer can permit a 2741 terminal operator to terminate the receiving of a message by pressing the ATTN (attention) key. To provide for such action, the programmer issues a

```
DFHTC TYPE=WRITE,                                              *
      WRBRK=symbolic address
```

macro instruction, where symbolic address is the label of a routine to which control is transferred if the terminal operator presses the ATTN

key while a message is being received. (Write Break support is not available under CICS/DOS/VS. For additional information, see "2741 Read Attention and Write Break Support.")

READ DATA FROM A TERMINAL (READ)

The application programmer can request that data be read from a terminal by issuing the

```
DFHTC TYPE=READ
```

macro instruction. Before this macro instruction is issued, the address of the TIOA can be placed into TCTTEDA.

If a TIOA is not provided by the application program, terminal control attempts to use an existing TIOA attached to the TCTTE. If no TIOA is attached, terminal control acquires a new TIOA. If the length of the existing TIOA or of the TIOA provided by the application program is not adequate, or if other conditions exist that make the TIOA unusable, terminal control acquires a new TIOA for the read. When terminal control acquires a new TIOA, the previous TIOA is freed, and the application program must place the value contained at TCTTEDA into TIOABAR following completion of the read to ensure addressability to the correct TIOA.

A new TIOA is acquired by terminal control for the read when the

```
DFHTC TYPE=(READ,SAVE)
```

macro instruction is issued. All TIOAs currently chained off the TCTTE are retained and may be subsequently reused; a new TIOA is dynamically acquired for this read (according to the length specified in the TCTLE) and is added to the chain.

Upon completion of a (REAL,SAVE), the application programmer must place the value contained at TCTTEDA into TIOABAR to establish addressability for the newly acquired TIOA. The number of bytes read is provided by CICS/VS at TICATDI.

A write followed by a read operation can be specified in a single request as discussed under "Write Data to a Terminal (WRITE)."

When input is to be received from a terminal of the 3270 Information Display System, the application programmer can use

```
DFHTC TYPE=(READ,TEXT)
```

or

```
DFHTC TYPE=READ
DFHTC TYPE=TEXT
DFHTC TYPE=WAIT
```

to request a temporary override of the uppercase translation features of CICS/VS, thus allowing a message containing both uppercase and lowercase data to be received from a terminal.

If 2741 read attention support is generated into CICS/VS, the application programmer can permit a 2741 terminal operator to signal Read Attention by pressing the ATTN key after typing a message. To provide for such action, the programmer issues a

```
DFHTC TYPE=READ,                                        *
      RDATT=symbolic address
```

macro instruction, where symbolic address is the label of a routine to which control is returned if the terminal operator terminates the input by pressing the ATTN key. (See "2741 Read Attention and Write Break Support" in Chapter 11.)

SYNCHRONIZE TERMINAL INPUT/OUTPUT FOR A TRANSACTION (WAIT)

In a task under which more than one terminal operation is performed, the application programmer must ensure that a current terminal operation is complete before another begins. This can be done by issuing the

```
DFHTC TYPE=WAIT
```

macro instruction, where the WAIT parameter is coded separately, as shown, or in combination with READ and/or WRITE. A PUT can be coded in place of a (WRITE,WAIT); a GET can be coded in place of a (READ,WAIT). A wait should be issued for each read request to ensure that the data has been transferred into the TIOA.

A wait may cause execution of a task to be suspended. If suspension is necessary, control is returned to CICS/VS. Execution of the task is resumed after the write and/or read is posted complete.

A wait need not be coded for a write if the write is the last terminal operation of the transaction. The TIOA is retained until it is written, even if the transaction and its associated storage are deleted from the system before the write occurs.

CONVERSE WITH A TERMINAL (CONVERSE)

The application programmer can request a conversational mode of communication with the terminal by issuing the

```
DFHTC TYPE=CONVERSE
```

macro instruction, where CONVERSE (or CONV) is the same as (WRITE,READ,WAIT). The execution of this instruction is always in the sequence: WRITE, implied wait, READ, WAIT. In the case of a 3270 terminal (operating in 2260 compatibility mode), the screen is not erased and writing begins wherever the cursor is located when this macro instruction is issued.

DISCONNECT A SWITCHED LINE (DISCONNECT)

The application programmer can use the

```
DFHTC TYPE=DISCONNECT
```

macro instruction to break a line connection between a terminal and a computer; this applies only to switched lines.

EXAMPLES

The following examples show the coding required to (1) acquire storage for use as a terminal input/output area through the DFHSC macro instruction, (2) place the address of the acquired area into TCTTEDA, (3) place the length of the data to be written into TIOATDL, (4) issue a terminal control macro instruction to a 3270 terminal, causing erasing of the screen, returning of the cursor to the upper left corner of the screen, writing to the terminal, and reading from the terminal (allowing

terminal control to manage storage for the TIOA), and (5) establish addressability to the TIOA into which the data was read.

For Assembler language:

```
        L     TCTTEAR,TCAFCAAA      ESTABLISH ADDRESSABILITY FOR TCTTE
        DFHSC TYPE=GETMAIN,         OBTAIN TIOA FOR OUTPUT DATA          *
              NUMBYTE=80,                                                *
              CLASS=TERMINAL
        L     TIOABAR,TCASCSA       ADDRESS OF TIOA
        ST    TIOABAR,TCTTEDA       PLACE OUTPUT ADDRESS IN TCTTE
        MVC   TIOADBA(80),DATA      PLACE DATA IN TIOA
        MVC   TIOATDL,=H'80'        PLACE DATA LENGTH IN TIOATDL
              .
              .
              .
        DFHTC TYPE=(WRITE,ERASE,    ISSUE WRITE TO TERMINAL              *
              READ,WAIT)            ERASE BEFORE WRITE, THEN READ
        L     TIOABAR,TCTTEDA       ESTABLISH ADDRESSABILITY FOR TIOA
```

For ANS COBOL:

```
        MOVE TCAFCAAA TO TCTTEAR.   NOTE EST ADDRESSABILITY FOR TCTTE.
        DFHSC TYPE=GETMAIN,         OBTAIN TIOA FOR OUTPUT DATA          *
              NUMBYTE=80,                                                *
              CLASS=TERMINAL
        MOVE TCASCSA TO TIOABAR.    NOTE ADDRESS OF TIOA.
        MOVE TIOABAR TO TCTTEDA.    NOTE PLACE ADDR OF TIOA IN TCTTE.
        MOVE DATA TO TIOADATA.      NOTE PLACE DATA IN TIOA (TIOADATA IS
                                    USER-DEFINED).
        MOVE 80 TO TIOATDL.         NOTE PLACE DATA LENGTH IN TIOATDL.
              .
              .
              .
        DFHTC TYPE=(WRITE,ERASE,    ISSUE WRITE TO TERMINAL              *
              READ,WAIT)            ERASE BEFORE WRITE, THEN READ
        MOVE TCTTEDA TO TIOABAR.    NOTE EST ADDRESSABILITY FOR TIOA.
```

For PL/I:

```
        TCTTEAR=TCAFCAAA;           /*EST ADDRESSABILITY FOR TCTTE*/
        DFHSC TYPE=GETMAIN,         OBTAIN TIOA FOR OUTPUT DATA          *
              NUMBYTE=80,                                                *
              CLASS=TERMINAL
        TIOABAR=TCASCSA;            /*ADDRESS OF TIOA*/
        TCTTEDA=TIOABAR;            /*PLACE ADDR OF TIOA IN TCTTE*/
        TIOADATA=DATA;              /*PLACE DATA IN TIOA (TIOADATA IS
                                    USER-DEFINED)*/
        TIOATDL=80;                 /*PLACE DATA LENGTH IN TIOATDL*/
              .
              .
              .
        DFHTC TYPE=(WRITE,ERASE,    ISSUE WRITE TO TERMINAL              *
              READ,WAIT)            ERASE BEFORE WRITE, THEN READ
        TIOABAR=TCTTEDA;            /*EST ADDRESSABILITY FOR TIOA*/
```

FILE SERVICES

File management provides the capability, through the file control program, to process fixed- or variable-length, blocked or unblocked, undefined, or segmented records of a direct access data set.

File control uses standard access methods of the host operating system (OS/VS or DOS/VS), namely:

- Direct Access Method (DAM)
- Indexed Sequential Access Method (ISAM)
- Virtual Storage Access Method (VSAM)

All data sets to be processed by file control must be described in the file control table (FCT). Application programs can access DAM data sets on a logical record level, deblocking services being provided by file control. If an ISAM data set is converted to a VSAM data set organization, using VSAM data set conversion utilities, no alteration to application programs that access the data set is necessary, but the file control table must be changed.

Through the file control macro instruction (DFHFC), an application program can perform file inquiry, which means, in effect, read a record from an existing data set on a direct access storage device; update an existing record in a data set; or add a new record to a data set. That program can also obtain a main storage area to create a new record or release a main storage area that has been acquired. File management also provides a single or grcup record deletion capability exclusively for VSAM key-sequenced data sets.

The application program can browse a data set. This browsing is comparable to a visual, sequential search of a file. File control macro instructions are used to specify a starting point for the browse, request each succeeding record, reset the starting point for the browse (if desired), and terminate the browse.

General file-handling capabilities available to application programs include indirect access to data sets, handling of "duplicates" data sets, and use of segmented records. The application programmer must work with the system programmer responsible for setting up the file control table (FCT) when planning to use any of these capabilities. For additional information, see "Data Base Considerations" in Chapter 11.

Optional access to the Data Language/I (DL/I) facility of the IBM Information Management System/Virtual Storage (IMS/VS, Program Product 5740-XX2) is provided under CICS/OS/VS. CICS/DOS/VS interfaces to DL/I DOS/VS (Program Product 5746-XX1). See Chapter 12 for information concerning the use of DL/I in a CICS/VS application program.

All buffers and work areas needed for data set operations are acquired by file control in accordance with the data set descriptions supplied in the FCT by the system programmer. All data sets and all segment sets referenced in DFHFC macro instructions must have been defined in the FCT. The application programmer may work with the system programmer in setting up these data set descriptions. However, the application program need be concerned only with logical records; it is not directly involved with other characteristics of the data set.

Note that while file management supports the user's data base, transient data management supports sequential data sets.

An application program always operates on DAM or ISAM data in one of two main storage areas: (1) a file input/output area (FIOA) or (2) a file work area (FWA). A FIOA is required to handle records that are read-only, unsegmented, and unblocked. A FWA is required to handle records that are new, segmented, blocked, or to be updated. In addition, a FWA is always used in a browse operation. For a VSAM data set, all data is read into or written from a FWA with one exception: a locate mode read-only (inquiry) request is entered for a nonsegmented record. In this case, the retrieved record is not moved to a FWA. A VSAM work area (VSWA) is established by CICS/VS, and the address of

the retrieved record, as it is positioned in the VSAM buffer, is made available to the application program at VSWAREA within the VSWA. The application programmer must provide a symbolic storage definition for this area (for example, an Assembler-language DSECT) and establish addressability to it. The record remains in the VSAM buffer, and care must be taken to ensure that it is not modified. It is possible for the user to determine which area (FIOA, FWA, or VSWA) is returned in response to a file request. Refer to the specific area under "Storage Definitions" for details.

Note: CICS/VS permits the sharing of VSAM resources. Resources to be shared are identified in the DFHFCT TYPE=SHRCTL macro instruction as explained in the CICS/VS System Programmer's Reference Manual. When a task requires resources in several VSAM data sets at the same time and these data sets are sharing resources, the possibility of a lockout increases.

File control executes at the priority of the requesting program, under control of the TCA of the requesting program, saving and restoring registers from this TCA. The application programmer can check the CICS/VS response to a request for file services as explained under "Test Response to a Request for File Services." Control can be routed to any of various user-written exception-handling routines based on the outcome of the file operation.

The application programmer must specify parameter values when using the file control macro instruction, in either of two ways:

- By including the parameters in operands of the DFHFC macro instruction by which file services are requested, or
- By coding instructions that place the parameter values in fields of the TCA prior to issuing the DFHFC macro instruction.

The second of these approaches provides flexibility in that the parameter values of a single DFHFC macro instruction can vary, to meet the logic needs of the application program.

File services that can be requested through the file control macro instruction are explained in the following paragraphs. Programming considerations related to each type of request (DFHFC TYPE=keyword operand) are discussed. For additional coding details, the application programmer should refer to "DFHFC Macro Instruction" in Chapter 7.

RANDOMLY RETRIEVE DATA FROM A DATA SET (GET)

The application programmer can randomly retrieve data from a data set by issuing the

```
DFHFC TYPE=GET,                                                *
      .
      .
      .
```

macro instruction. This macro instruction is used for random read-only (inquiry) or update (DFHFC TYPE=GET, TYPOPER=UPDATE) operations. The requested data record is returned in (1) a file input/output area (FIOA) for read-only operations with unsegmented, unblocked records from a DAM or ISAM data set; (2) a file work area (FWA) for update operations, read-only operations with segmented or blocked records, or for read-only operations with a VSAM data set; or (3) in a VSAM buffer area for locate mode read-only operations on nonsegmented records of a VSAM data set.

Before file services are requested in an application program by means of the DFHFC TYPE=GET macro instruction, the application programmer must provide instructions that symbolically define any required FIOA, FWA and/or VSWA by (1) copying the appropriate CICS/VS control section storage definitions (DFHFIOA, DFHFWADS, and/or DFHVSWA) provided by CICS/VS, and (2) providing his own storage definitions for the user's section of the FIOA, FWA, and/or the user's record in the VSAM buffer.

Note: Under CICS/OS/VS, if ISAM data is to be placed in a FIOA, a 16-byte filler must be defined following the statement that copies DFHFIOA and ahead of the user's data definition.

CICS/VS performs the following services in response to a DFHFC TYPE=GET macro instruction:

1. Acquires the main storage area required to read a record
2. Reads the requested data
3. Locates the requested logical record

To communicate to CICS/VS the identity of the record required in an input/output operation, the application program must utilize a record identification field for the data set. The format of this field, as required for the various access methods, is described under "Record Identification Field" in Chapter 11. When a DAM data set is referenced, the desired record is accessed on the basis of the block reference field. An ISAM data set is referenced on the basis of the logical record key. For VSAM data sets, a record can be accessed with either a relative byte address or a logical record key. A search by key may be for a key exactly equal to the search key, or for one equal to or greater than the search key. A search may also be for a partial key (the first two bytes, or any number specified by the programmer), which may serve as a generic key. The generic or partial key search may, again, be either for an equal key or for an equal or greater key, but only the number of bytes specified will be compared. A protected, key-sequenced VSAM data set can be updated only on the basis of the full key equal search. Because CICS/VS application programs must have the quality of quasi-reentrance, the record identification field should be contained in a task-related storage area, but that area should not be within the application program.

In addition, CICS/VS can perform the following services, depending on the operands included in the DFHFC TYPE=GET macro instruction:

- Retrieve a record indirectly
- Segment a record for inquiry (read-only) and return the requested segments in a work area
- Acquire a file work area of the same length as the requested record when the record is to be updated or when records are blocked or segmented
- Unpack a segmented record into a work area of the same length as the requested record

When a nonsegmented record of a VSAM data set is accessed in response to a read-only request, the application programmer may specify either move mode or locate mode processing. Under move mode, the record is handled as any DAM or ISAM record. Under locate mode, the retrieved record is made available to the application program in the VSAM buffer. The application programmer must have copied the CICS/VS symbolic storage

definition for the VSAM work area (DFHVSWA) and must also provide a symbolic storage definition for the record that is retrieved.

After requesting file services, the programmer must establish addressability for any required FIOA or FWA. The address of the area involved, provided by CICS/VS at TCAFCAA, must be placed at FIOABAR or FWACBAR. When locate mode is used, the address of the VSAM work area (VSWA) is in TCAFCAA and must be placed in VSWABAR. The address of the area that holds the requested record is at VSWAREA within VSWA and

must be moved to the base locator that the programmer has established for the symbolic storage definition of the area.

When retrieving variable-length records from a VSAM data set in move mode, the file ccntrol program creates a conventional LLØØ length field and places it before the record in the FWA. In locate mode, the length is not included in the record itself but is placed at VSWALEN in the VSWA.

When a VSAM record is retrieved for update, VSAM maintains exclusive control of the control interval containing that record. If there is an attempt to retrieve another record from the same interval before the previous one is updated by a DFHTC TYPE=INPUT or before the operation is terminated by a DFHFC TYPE=RELEASE, a lockout condition may exist.

The application programmer should issue a DFHFC TYPE=RELEASE macro instruction to free an FIOA or FWA acquired in response to a request for file services. A DFHFC TYPE=RELEASE macro instruction should be issued to release a VSWA established for a VSAM read-only request using locate mode I/O. CICS/VS frees any of these areas that are not freed by the application program upon task termination.

Random Read-Only Retrieval

The following examples show how to do a random read-only (inquiry) operation on a record of master data set, assuming blocked or segmented records.

For Assembler language:

```
         COPY   DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEYF     DS     CL8                   RECORD IDENT FIELD IN TWA
FWACBAR  EQU    7                     ASSIGN BASE REGISTER FOR FWA
         COPY   DFHFWADS              SYMBOLICALLY DEFINE FWA
RECORD   DS     0CL350                RECORD LAYOUT FOLLOWS CONTROL
         .                            FIELD AND HAS SAME BASE REGISTER
         .
         .
         MVC    KEYF,ACCTNO           MOVE RECORD IDENT TO KEY FIELD
READREC  DFHFC  TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
                DATASET=MASTERA,                                          *
                RDIDADR=KEYF
         L      FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
```

For ANS COBOL:

```
    02  FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
        .                            NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.         NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PICTURE X(8).          NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.         NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PICTURE X(350).      NOTE DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
PROCEDURE DIVISION.
```

```
    MOVE CSACDTA TO TCACBAR.        NOTE ESTABLISH TCA ADDRESSABILITY.
         .
         .
         .
    MOVE ACCTNO TO KEYF.            NOTE MOVE RECORD IDENT TO KEY.
READREC.
         DFHFC TYPE=GET,            GET RECORD FROM MASTER DATA SET     *
               DATASET=MASTERA,                                         *
               RDIDADR=KEYF
    MOVE TCAFCAA TO FWACBAR.        NOTE ESTABLISH FWA ADDRESSABILITY.
```

For PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02  KEYF CHAR(8);               /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02  RECORD CHAR(350);           /*DEFINE RECORD LAYOUT IN FWA*/
        .
        .
        .
KEYF=ACCTNO;                        /*ASSIGN RECORD IDENT TO KEY FIELD*/
READREC:
        DFHFC TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,                                          *
              RDIDADR=KEYF
FWACBAR=TCAFCAA;                    /*ESTABLISH ADDRESSABILITY FOR FWA*/
```

The following examples show how to retrieve a record from a VSAM data set using locate mode I/O. If the record being retrieved is of variable length, the traditional LLØØ field will not be part of the record. The length of the record can be found in VSWALEN in the VSWA.

For Assembler language:

```
        COPY  DFHTCADS              COPY TCA SYMBOLIC STORAGE DEFN
KEYF    DS    CL8                   DEFINE KEY FIELD IN TWA
VSWABAR EQU   7                     ASSIGN BASE REGISTER FOR VSWA
RECBAR  EQU   8                     ASSIGN BASE REGISTER FOR RECORD
        COPY  DFHVSWA               COPY VSWA SYMBOLIC DEFN
RECDS   DSECT                       DUMMY SECTION FOR RECORD
        USING *,RECBAR              MAKE RECORD ADDRESSABLE
RECORD  DS    OCL350                DEFINE RECORD LAYOUT
        .
        .
        .
        MVC   KEYF,ACCTNO           MOVE RECORD ID TO KEY FIELD
READREC DFHFC TYPE=GET,             GET A RECORD FROM MASTER            *
              DATASET=MASTVSAM,        VSAM DATA SET USING              *
              RDIDADR=KEYF,            LOCATE MODE                      *
              MODE=LOCATE
        L     VSWABAR,TCAFCAA       ESTABLISH VSWA ADDRESSABILITY
        L     RECBAR,VSWAREA        ESTABLISH RECORD ADDRESSABILITY
        L     3,VSWALEN             LOAD RECORD LENGTH INTO WORK REG
```

For ANS COBOL:

```
    02  VSWABAR PICTURE S9(8) COMPUTATIONAL.
                                    NOTE DEFINE BASE REGISTER FOR VSWA.
    02  RECBAR PICTURE S9(8) COMPUTATIONAL.
        .                           NOTE DEFINE BASE REGISTER FOR RECORD.
        .
```

```
        .
01  DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PICTURE X(8).          NOTE DEFINE KEY FIELD IN TWA.
    02  RECLEN PICTURE S9(8) COMPUTATIONAL.
        .                            NOTE DEFINE RECORD LENGTH WORK AREA.
        .
        .
01  DFHVSWA COPY DFHVSWA.           NOTE COPY SYMBOLIC STRG DEFN FOR VSWA.
01  RECDS SYNCHRONIZED.             NOTE DEFINE SYMBOLIC STRG DEFN FOR RECORD.
    02  RECORD PICTURE X(350).      NOTE DEFINE RECORD LAYOUT.
        .
        .
        .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACEAR.         NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
    MOVE ACCTNO TO KEYF.             NOTE MOVE RECORD ID TO KEY FIELD.
READREC.
        DFHFC TYPE=GET,              GET A RECORD FROM MASTER            *
              DATASET=MASTVSAM,          VSAM DATA SET USING           *
              RDIDADR=KEYF,              LOCATE MODE                   *
              MODE=LOCATE
    MOVE TCAFCAA TO VSWABAR.         NOTE ESTABLISH VSWA ADDRESSABILITY.
    MOVE VSWAREA TO RECBAR.          NOTE ESTABLISH RECORD ADDRESSABILITY.
    MOVE VSWALEN TO RECLEN.          NOTE MOVE RECORD LENGTH TO WORK AREA.
```

For PL/I:

```
%INCLUDE DFHTCADS;                   /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02  KEYF CHAR(8),                /*DEFINE KEY FIELD IN TWA*/
    02  RECLEN FIXED BINARY(31);     /*DEFINE RECORD LENGTH WORK AREA*/
%INCLUDE DFHVSWA;                    /*COPY SYMBOLIC STRG DEFN FOR VSWA*/
DEFINE 01 RECDS BASED (RECBAR),      /*DEFINE SYMBOLIC STRG DEFN FOR RECORD*/
    02  RECORD CHAR(350);            /*DEFINE RECORD LAYOUT*/
        .
        .
        .
KEYF=ACCTNO;                         /*MOVE RECORD ID TO KEY FIELD*/
READREC:
        DFHFC TYPE=GET,              GET A RECORD FROM MASTER            *
              DATASET=MASTVASM,          VSAM DATA SET USING           *
              RDIDADR=KEYF,              LOCATE MODE                   *
              MODE=LOCATE
VSWABAR=TCAFCAA;                     /*ESTABLISH ADDRESSABILITY FOR VSWA*/
RECBAR=VSWAREA;                      /*ESTABLISH ADDRESSABILITY FOR RECORD*/
RECLEN=VSWALEN;                      /*MOVE RECORD LENGTH TO WORK AREA*/
```

Random Retrieval for Update

The following examples show how to randomly retrieve a record on the master data set for update.

For Assembler language:

```
        COPY  DFHTCADS               COPY TCA SYMBOLIC STRG DEFN
KEYF    DS    CL8                    DEFINE KEY FIELD IN TWA
FWACBAR EQU   7                      ASSIGN BASE REGISTER FOR FWA
        COPY  DFHFWADS               SYMBOLICALLY DEFINE FWA
RECORD  DS    0CL350                 RECORD LAYOUT FOLLOWS CONTROL
        .                            FIELD AND HAS SAME BASE REGISTER
```

```
         .
         .
         MVC    KEYF,ACCTNC          MOVE RECORD IDENT TO KEY FIELD
READREC  DFHFC  TYPE=GET,            GET RECORD FROM MASTER DATA SET     *
                DATASET=MASTERA,     FOR UPDATE                          *
                RDIDADR=KEYF,                                            *
                TYPOPER=UPDATE
         L      FWACBAR,TCAFCAA      ESTABLISH ADDRESSABILITY FOR FWA
```

For ANS COBOL:

```
    02   FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
         .                          NOTE DEFINE BASE REGISTER FOR FWA.
         .
         .
01  DFHTCADS COPY DFHTCADS.        NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02   KEYF PICTURE X(8).        NOTE DEFINE KEY FIELD IN TWA.
         .
         .
         .
01  DFHFWADS COPY DFHFWADS.        NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02   RECORD PICTURE X(350).    NOTE DEFINE RECORD LAYOUT IN FWA.
         .
         .
         .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.       NOTE ESTABLISH TCA ADDRESSABILITY.
         .
         .
         .
    MOVE ACCTNO TO KEYF.           NOTE MOVE RECORD IDENT TO KEY.
READREC.
         DFHFC TYPE=GET,           GET RECORD FROM MASTER DATA SET     *
               DATASET=MASTERA,                                        *
               RDIDADR=KEYF,                                           *
               TYPOPER=UPDATE
    MOVE TCAFCAA TO FWACBAR.       NOTE ESTABLISH FWA ADDRESSABILITY.
```

For PL/I:

```
%INCLUDE DFHTCADS;                 /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02 KEYF CHAR(8);               /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                 /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02 RECORD CHAR(350);           /*DEFINE RECORD LAYOUT IN FWA*/
         .
         .
         .
KEYF=ACCTNO;                       /*ASSIGN RECORD IDENT TO KEY FIELD*/
READREC:
       DFHFC TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
             DATASET=MASTERA,                                          *
             RDIDADR=KEYF,                                             *
             TYPOPER=UPDATE
FWACBAR=TCAFCAA;                   /*ESTABLISH ADDRESSABILITY FOR FWA*/
```

Random Retrieval through Indirect Access

The following examples show how to randomly retrieve a record for update when the key for the desired record is unknown. A cross-index data set containing the master key is available, making it possible to access the record indirectly. (See "Indirect Accessing" in Chapter 11 to learn more about this capability.)

For Assembler language:

```
         COPY   DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEYF     DS     CL25                  DEFINE KEY FIELD IN TWA
FWACBAR  EQU    7                     ASSIGN BASE REGISTER FOR FWA
         COPY   DFHFWADS              SYMBOLICALLY DEFINE FWA
RECORD   DS     0CL350                RECORD LAYOUT FOLLOWS CONTROL
         .                            FIELD AND HAS SAME BASE REGISTER
         .
         MVC    KEYF,INDEXA           MOVE INDEX IDENT TO KEY FIELD
READING  DFHFC  TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
                DATASET=MASTERA,      BY FIRST ACCESSING A CROSS-INDEX    *
                RDIDADR=KEYF,         DATA SET NAMED INDIRECT             *
                TYPOPER=UPDATE,                                           *
                INDEX=INDIRECT
         L      FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
```

For ANS COBOL:

```
    02   FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
         .                           NOTE DEFINE BASE REGISTER.
         .
01  DFHTCADS COPY DFHTCADS.         NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02   KEYF PICTURE X(25).        NOTE DEFINE KEY FIELD IN TWA.
         .
         .
01  DFHFWADS COPY DFHFWADS.         NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02   RECORD PICTURE X(350).     NOTE DEFINE RECORD LAYOUT IN FWA.
         .
         .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.        NOTE ESTABLISH TCA ADDRESSABILITY.
         .
         .
    MOVE PARTNAME TO KEYF.          NOTE MOVE INDEX IDENT TO KEY.
READREC.
       DFHFC TYPE=GET,              GET RECORD FROM MASTER DATA SET     *
             DATASET=MASTERA,       BY FIRST ACCESSING A CROSS-INDEX    *
             RDIDADR=KEYF,          DATA SET NAMED INDEXAB              *
             TYPOPER=UPDATE,                                            *
             INDEX=INDEXAB
    MOVE TCAFCAA TO FWACBAR.        NOTE ESTABLISH FWA ADDRESSABILITY.
```

For PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02 KEYF CHAR(25);               /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02 RECORD CHAR(350);            /*DEFINE RECORD LAYOUT IN FWA*/
       .
       .
KEYF=PARTNAME;                      /*ASSIGN INDEX IDENT TO KEY FIELD*/
READREC:
       DFHFC TYPE=GET,              GET RECORD FROM MASTER DATA SET     *
             DATASET=MASTERA,       BY FIRST ACCESSING A CROSS-INDEX    *
             RDIDADR=KEYF,          DATA SET NAMED INDEXAB              *
             TYPOPER=UPDATE,                                            *
             INDEX=INDEXAB
FWACBAR=TCAFCAA;                    /*ESTABLISH ADDRESSABILITY FOR FWA*/
```

RANDOMLY UPDATE OR ADD DATA TO A DATA SET (PUT)

The application programmer can randomly update or add data to a data set by issuing the

```
DFHFC TYPE=PUT,                                                    *
      .
      .
      .
```

macro instruction. This macro instruction is used to (1) update an existing record that has been retrieved through the DFHFC TYPE=GET,TYPOPER=UPDATE macro instruction, (2) add a new record to an existing data set, or (3) update an existing record in a nonkeyed DAM data set without first reading the record for update. A DFHFC TYPE=PUT macro instruction must never be issued without first issuing a DFHFC TYPE=GET, TYPOPER=UPDATE or DFHFC TYPE=GETAREA macro instruction, because the results of such action are unpredictable.

When a VSAM key-sequenced data set is being processed, the application programmer also has a deletion capability. A DFHFC TYPE=PUT,TYPOPER=DELETE macro instruction can be used to delete a record previously retrieved by a DFHFC TYPE=GET,TYPOPER=UPDATE macro instruction.

A file work area (FWA) is used to contain the record or segments to be written or updated. The first 16 bytes of this work area are the CICS/VS control section, which is followed by the actual record or segments to be written to a data set.

CICS/VS performs the following services in response to a DFHFC TYPE=PUT macro instruction:

1. Writes updated or new records on user-defined data sets
2. Acquires or locates the main storage and control blocks required to write the record
3. Releases all data set storage associated with the request to write

- Packs a segmented record, depending on the data set organization and the operands included in the DFHFC TYPE=PUT macro instruction

Before file services can be requested by means of the DFHFC TYPE=PUT macro instruction, the application programmer must provide instructions that do the following:

1. Symbolically define the FWA by (1) copying the appropriate CICS/VS control section storage definition (DFHFWADS), and (2) providing a storage definition for the user's section of the FWA.
2. Establish addressability for the new FWA by specifying a symbolic base address for the FWA.
3. Place the address of the FWA in TCAFCAA. (This address was made available to the application program by CICS/VS in response to the DFHFC TYPE=GET or DFHFC TYPE=GETAREA request by which the FWA was acquired. It must have been stored by the application program at that time, and should be moved to TCAFCAA immediately preceding the DFHFC TYPE=PUT request, with no intervening requests that could cause the contents of TCAFCAA to be altered.)

If the records being written to a data set are undefined, the application programmer must place the length of the record being written in TCAFCURL.

For records written to a variable-length VSAM data set, the length of the record should be placed in an LLØØ field in the beginning of the record. This field is used by CICS/VS to determine the length of the record and is not written to the data set.

VSAM does not allow addition of a record to a control interval from which a record has already been retrieved for update. If an application program attempts to add a record to such a control interval before the previous record is updated by a DFHFC TYPE=PUT or the update is terminated by a DFHFC TYPE=RELEASE, a lockout condition exists.

The programmer who is adding records to a DAM data set should also refer to "Adding Records to DAM Data Sets" in Chapter 11.

The following examples show how to randomly retrieve a record for updating and then return that record to the data set.

For Assembler language:

```
         COPY   DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEYF     DS     CL8                   DEFINE KEY FIELD IN TWA
FWACBAR  EQU    7                     ASSIGN BASE REGISTER FOR FWA
         COPY   DFHFWADS              SYMBOLICALLY DEFINE FWA
RECORD   DS     0CL350                RECORD LAYOUT FOLLOWS CONTROL
         .                            FIELD AND HAS SAME BASE REGISTER
         .
         .
READUPD  DFHFC  TYPE=GET,             PEAD RECORD FOR UPDATE            *
                DATASET=MASTERB,                                        *
                RDIDADR=KEYF,                                           *
                TYPOPER=UPDATE
         L      FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
         .
         .        (update record)
         .
         ST     FWACBAR,TCAFCAA       PIACE FWA ADDRESS IN TCA
WRITEUP  DFHFC  TYPE=PUT              WRITE THE UPDATED RECORD
```

For ANS COBOL:

```
    02  FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
        .                            NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.         NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PICTURE X(8).          NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.         NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PICTURE X(350).      NOTE DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.         NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
```

```
READUPD.
       DFHFC TYPE=GET,              READ RECORD FOR UPDATE              *
             DATASET=MASTERB,                                           *
             RDIDADR=KEYF,                                              *
             TYPOPER=UPDATE
    MOVE TCAFCAA TO FWACBAR.        NOTE ESTABLISH FWA ADDRESSABILITY.
         .
         .  (update record)
         .
    MOVE FWACBAR TO TCAFCAA.        NOTE MOVE ADDRESS OF FWA TO TCA.
WRITEUP.
       DFHFC TYPE=PUT               WRITE THE UPDATED RECORD
```

For PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02 KEYF CHAR(8);                /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02 RECORD CHAR(350);            /*DEFINE RECORD LAYOUT IN FWA*/
         .
         .
READUPD:
       DFHFC TYPE=GET,              READ RECORD FOR UPDATE              *
             DATASET=MASTERB,                                           *
             RDIDADR=KEYF,                                              *
             TYPOPER=UPDATE
FWACBAR=TCAFCAA;                    /*ESTABLISH ADDRESSABILITY FOR FWA*/
         .
         .  (update record)
         .
TCAFCAA=FWACBAR;                    /*PLACE ADDR OF WORK AREA IN TCA*/
WRITEUP:
       DFHFC TYPE=PUT               WRITE THE UPDATED RECORD
```

RANDOMLY DELETE DATA FROM A DATA SET (DELETE)

When processing a VSAM key-sequenced data set, the application programmer can delete single records or groups of records at random from the data set. Single records can be deleted in either of two ways: (1) by specifying TYPOPER=DELETE in a PUT macro instruction for a record previously retrieved for update (see the preceding discussion of PUT), or (2) by issuing a DFHFC TYPE=DELETE macro instruction. To delete groups of records, the seccnd of these methods, a

```
       DFHFC TYPE=DELETE,                                               *
             .
             .
             .
```

macro instruction must be used.

When a single record is to be deleted by means of a DFHFC TYPE=DELETE macro instruction, that record can be searched for by key or by relative byte address. When the record is accessed by key, the complete key can be used as a search argument, or only a partial key (the length of which is specified by the application programmer) can be used. In the latter case, all records whose keys begin with the search argument are deleted. CICS/VS returns a count of the number of records deleted at TCAFCNRD.

Neither a FIOA nor a FWA is required for a delete operation.

OBTAIN A FILE WORK AREA (GETAREA)

The application programmer can obtain an area of main storage in which to create a new record for a data set by issuing the

```
DFHFC TYPE=GETAREA,                                          *
      .
      .
```

macro instruction. The new main storage area is a file work area (FWA) and can be obtained only through a DFHFC TYPE=GETAREA request. (A storage control DFHSC TYPE=GETMAIN request cannot be used for file operations.)

CICS/VS performs the following services in response to a DFHFC TYPE=GETAREA macro instruction:

1. Acquires main storage (an FWA) for the creation of a new record
2. Includes and initializes the FWA control fields (a 16-byte prefix to the FWA) required by file control

If the application programmer intends to add several new logical records whose keys are in ascending sequence to a VSAM data set, he can write a DFHFC TYPE=GETAREA,TYPOPER=MASSINSERT macro instruction. In this case, the FWA is retained and made available to the application program after each DFHFC TYPE=PUT macro instruction adding a record to the data set. If storage initialization is specified in the DFHFC TYPE=GETAREA macro instruction, the FWA is reinitialized before each return to the application program. A mass insert operation is terminated by issuing a DFHFC TYPE=RELEASE macro instruction.

When the DFHFC TYPE=GETAREA macro instruction is used, the application programmer must provide instructions that do the following:

- Symbolically define the FWA by (1) copying the appropriate CICS/VS control section storage definition (DFHWADS), and (2) providing a storage definition for the user's section of the FWA.
- Establish addressability for the new FWA by specifying a symbolic base address for the FWA. (The address of the area involved, returned by CICS/VS at TCAFCAA, must be placed in FWACBAR.)

The following examples show how to obtain a FWA, build a new record in the FWA, and then write the record to a data set.

For Assembler language:

```
        COPY   DFHTCADS             COPY TCA SYMBOLIC STRG DEFN
KEYF    DS     CL8                  DEFINE KEY FIELD IN TWA
FWACBAR EQU    7                    ASSIGN BASE REGISTER FOR FWA
        COPY   DFHFWADS             SYMBOLICALLY DEFINE FWA
RECORD  DS     0CL350               RECORD LAYOUT FOLLOWS CONTROL
        .                           FIELD AND HAS SAME BASE REGISTER
        .
        .
NEWREC  DFHFC  TYPE=GETAREA,        OBTAIN A FWA TO CREATE A NEW      *
               DATASET=MASTERC      RECORD FOR A DATA SET
        L      FWACBAR,TCAFCAA      ESTABLISH ADDRESSABILITY FOR FWA
        .
        .      (build new record)
        .
        ST     FWACBAR,TCAFCAA      PLACE ADDR OF NEW RECORD IN TCA
WRITNEW DFHFC  TYPE=PUT,            WRITE THE NEW RECORD              *
```

```
            TYPOPER=NEWREC,                                          *
            RDIDADR=KEYF
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                             NOTE DEFINE BASE REGISTER FOR FWA.
      .
      .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PICTURE X(8).           NOTE DEFINE KEY FIELD IN TWA.
      .
      .
      .
01 DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE X(350).        NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.         NOTE ESTABLISH TCA ADDRESSABILITY.
      .
      .
      .
NEWREC.
       DFHFC TYPE=GETAREA,          OBTAIN A FWA TO CREATE A NEW        *
             DATASET=MASTERC        RECORD FOR A DATA SET
   MOVE TCAFCAA TO FWACBAR.         NOTE ESTABLISH FWA ADDRESSABILITY.
      .
      .    (build new record)
      .
   MOVE FWACBAR TO TCAFCAA.         NOTE ADDRESS OF NEW RECORD TO TCA.
WRITNEW.
        DFHFC TYPE=PUT,             WRITE THE NEW RECORD                *
              TYPOPER=NEWREC,                                           *
              RDIDADR=KEYF
```

For PL/I:

```
%INCLUDE DFHTCADS;                 /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF CHAR(8);                /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                 /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);            /*DEFINE RECORD LAYOUT IN FWA*/
      .
      .
NEWREC:
        DFHFC TYPE=GETAREA,        OBTAIN A FWA TO CREATE A NEW        *
              DATASET=MASTERC      RECORD FOR A DATA SET
FWACBAR=TCAFCAA;                   /*ESTABLISH ADDRESSABILITY FOR FWA*/
      .
      .    (build new record)
      .
TCAFCAA=FWACBAR;                   /*PLACE ADDR OF NEW RECORD IN TCA*/
WRITNEW:
        DFHFC TYPE=PUT,            WRITE THE NEW RECORD                *
              TYPOPER=NEWREC,                                          *
              RDIDADR=KEYF
```

RELEASE FILE STORAGE (RELEASE)

The application programmer can release the storage areas acquired for file control operations by issuing the

```
DFHFC TYPE=RELEASE,                                              *
              .
              .
              .
```

macro instruction. This macro instruction is used when (1) a record has been retrieved for update, (2) it is determined that the update should not occur, and (3) it is desired to release all encumbrances associated with the update operation (these may include FWA, FIOA, and exclusive control, which prevents other application programs from accessing the data to be updated). A DFHFC TYPE=RELEASE macro instruction is also used to release the storage occupied by a FWA or FIOA used for a read operation.

Before the DFHFC TYPE=RELEASE macro instruction is executed, the application programmer must ensure that the address of the FWA, FIOA, or VSWA to be released has been placed at TCAFCAA. Any associated areas are also released.

The application programmer should not expect to find an FIOA address at TCAFCAA in the case of a DSIDER, SEGIDER, INVREQ, or NOTOPEN condition, because CICS/VS does not acquire an FIOA if one of these conditions occurs.

A mass insert operation on a VSAM data set (initiated by a DFHFC TYPE=GETAREA,TYPOPER=MASSINSERT macro instruction, which is followed by DFHFC TYPE=PUT,TYPOPER=NEWREC macro instructions referring to the data set) is terminated by a DFHFC TYPE=RELEASE macro instruction. A DFHFC TYPE=RELEASE macro instruction should also be used to release the VSWA established by CICS/VS in response to a read-only request for a VSAM data set record retrieved under locate mode. Failure to release the VSWA may cause significant performance degradation or task suspension if subsequent accesses are made to the file.

The DFHFC TYPE=RELEASE macro instruction should not be specified if the DFHFC TYPE=PUT,TYPOPER=UPDATE macro instruction is used to perform a successful write of an updated record back to a data set. CICS/VS automatically releases all storage associated with the write operation. If an error condition occurs, preventing successful completion of the write, however, a DFHFC TYPE=RELEASE macro instruction should be issued to release the storage. DFHFC TYPE=RELEASE must be specified if a DUPREC (duplicate record) indication is returned in response to a DFHFC TYPE=PUT macro instruction to an ISAM data set. Refer to the description of the DUPREC operand of the DFHTC TYPE=CHECK macro instruction in Chapter 7 for more information on DUPREC.

CICS/VS performs the following services in response to a DFHFC TYPE=RELEASE macro instruction:

- Releases a FWA, FIOA, and/or VSWA
- Releases exclusive control of a record retrieved for update (if applicable)

Any FWAs, FIOAs, and VSWAs acquired during execution of a task are automatically released at termination of the task, if not released earlier in response to to a DFHFC TYPE=RELEASE macro instruction.

The following examples show how to request the release of a FWA.

For Assembler language:

```
FWACBAR EQU   7                     ASSIGN BASE REGISTER FOR FWA
        COPY  DFHFWADS              SYMBOLICALLY DEFINE FWA
RECORD  DS    0CL350                RECORD LAYOUT FOLLOWS CONTROL
        .                           FIELD AND HAS SAME BASE REGISTER
        .
        .
        ST    FWACBAR,TCAFCAA       ADDRESS OF FWA TO BE RELEASED
RLSEREC DFHFC TYPE=RELEASE          IN TCA AND ISSUE RELEASE REQUEST
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                             NOTE DEFINE BASE REGISTER FOR FWA.
      .
      .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE X(350).        NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.         NOTE ESTABLISH TCA ADDRESSABILITY.
      .
      .
      .
   MOVE FWACBAR TO TCAFCAA.         NOTE ADDR OF FWA TO BE RELEASED.
RLSEREC.
        DFHFC TYPE=RELEASE          ISSUE RELEASE REQUEST
```

For PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
      .
      .
      .
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);             /*DEFINE RECORD LAYOUT IN FWA*/
      .
      .
      .
TCACBAR=CSACDTA;                    /*ESTABLISH ADDRESSABILITY FOR TCA*/
      .
      .
      .
TCAFCAA=FWACBAR;                    /*ADDRESS OF FWA TO BE RELEASED*/
RLSEREC:
        DFHFC TYPE=RELEASE          ISSUE RELEASE REQUEST
```

BROWSING

Four DFHFC macro instructions are designed to facilitate browse (sequential search) operations against a direct access data set. The TYPE operand entries for these macro instructions are SETL, GETNEXT, RESETL, and ESETL. The capabilities associated with each are summarized below. Keyword operands to request checking of a CICS/VS response can be specified with these macro instructions as with other DFHFC macro instructions (see "Test Response to a Request for File Services"). Specific operands are discussed in detail, for each macro instruction, in Chapter 7.

When accessing a VSAM data set, the browse facility can be used to perform random skip-sequential processing. The following steps are required:

1. Group several random requests into ascending key sequence.

2. Issue a DFHFC TYPE=SETL macro instruction which positions to the first required record. To achieve this, the record identification field pointed to by the RDIDADR operand should be initialized to the key of the required record.

3. Prior to each DFHFC TYPE=GETNEXT macro instruction, place the key of the next required record into the record identification field.

This procedure allows quick random access to a VSAM data set by reducing index search time. When the record having the highest key has been retrieved, an ESETL or RESETL should be issued to terminate or reset the operation.

Initiate Sequential Retrieval (SETL)

The application programmer initiates a browse operation on a data set by issuing the

```
DFHFC TYPE=SETL,                                                    *
      .
      .
      .
```

macro instruction. This macro instruction is used to establish the position within the data set where the browse operation is to begin. It must be issued before any DFHFC TYPE=GETNEXT macro instruction. No data is available until a DFHFC TYPE=GETNEXT is used.

Recall that an application program communicates to CICS/VS the identity of a specific record required in an input/output operation by means of a record identification field established for the data set. (See "Record Identification Field" in Chapter 11.) For an ISAM data set, the browse operation begins at the first record with a key equal to or greater than the key provided in the record identification field for the data set. This key may be either a specific (complete) key or a generic (partial) key. For example, a complete key of D642BR17 causes sequential processing to begin at the first record with a key equal to or greater than that key. A generic key is one in which the application programmer supplies only the significant characters of a desired group of keys, padding the remainder of the key field with blanks or binary zeros. A generic key of D6420000 causes sequential processing to begin at the first record with a key whose first four characters are equal to or greater than D642xxxx, regardless of the character represented by each x. A key field of all binary zeros causes sequential processing to begin at the first logical record of the data set.

For a DAM data set, the record identification field must contain a specific block reference (for example, TTR or MBBCCHHR as explained in Chapter 11) which conforms to the addressing method defined for that data set. Processing begins with the specified block and continues with each subsequent block until the browse operation is terminated. If the DAM data set contains blocked records, processing begins at the first logical record of the first block and continues with each subsequent logical record.

For a VSAM data set, the contents of the record identification field may be either a key or a relative byte address. If the field contains a relative byte address, the browse operation begins at the location identified. If the field cortains a key, it may be either specific or generic. If the key is generic, the length of the partial key is specified in the first byte of the record identification field. In

either case, the application programmer can specify that the browse operation is to begin at the first record having a key (1) equal to the key in the record identification field (for generic, compared on only the number of bytes specified), or (2) equal to or greater than the key in the record identification field (again, for generic, compared on only the bytes specified).

When the DFHFC TYPE=SETL macro instruction is used, the application programmer must provide instructions that do the following:

- Symbolically define the FWA by (1) copying the appropriate CICS/VS control section storage definition (DFHFWADS), and (2) providing his own storage definition for the user's section of the FWA.

- Establish addressability for the FWA by specifying a symbolic base address for the FWA, typically following the DFHFC macro instruction. (The address of the FWA, provided by CICS/VS at TCAFCAA, must be placed at FWACBAR upon normal return from execution of the SETL macro instruction.)

In most cases, records retrieved during a browse operation are returned to the application program in a file work area (FWA). In locate mode the addresses of the logical record are passed in the VSWA. The FWA allocated by CICS/VS following a SETL request is unique for the duration of that particular browse operation. If the application program issues another SETL request, for the same or another data set, a different FWA is created by CICS/VS. Thus it is possible for a single application program to concurrently browse the same data set at several different locations.

Note that during a browse operation on a segmented data set, the original FWA (that is, the one allocated by CICS/VS in response to the SETL request) may be replaced with a different FWA if a segment set specified in a GETNEXT request requires a larger FWA than the segment set specified in the SETL request. In this situation, the application programmer should not assume that the same FWA is used for all GETNEXT requests. The address of the utilized FWA is available at TCAFCAA upon return from a GETNEXT request.

CICS/VS performs the following services in response to a DFHFC TYPE=SETL macro instruction:

1. Acquires the main storage I/O areas and work areas to be associated with this browse operation

2. Preserves the segment set name (if any) as the default segment set name to be used if none is specified in subsequent GETNEXT requests

3. Returns the address of the allocated FWA in TCAFCAA for other than locate-mode nonsegmented VSAM data set processing; returns the address of the allocated VSWA that will contain the VSAM buffer-area address of each retrieved record for locate-mode nonsegmented VSAM data set processing

The information supplied by the user in the record identification field is preserved by CICS/VS for use when GETNEXT requests are issued. Since CICS/VS places into this field the identification of each record retrieved in response to a subsequent GETNEXT request, the field should not be released by the application program.

The information placed into the record identification field by CICS/VS is always in a form which completely identifies the record. For example, assume a browse operation is to start with the first logical record of a blocked, keyed DAM data set. Before issuing the

DFHFC TYPE=SETL macro instruction, the application programmer should place the TTR (assuming that is the addressing method) of the first block into the Record Identification field. After executing each DFHFC TYPE=GETNEXT macro instruction, CICS/VS places the complete logical record identification into the record identification field. After the first GETNEXT, the record identification field might contain

X'00010504'

where 0001 represents the TTR value, 05 represents the block key, and 04 represents the record key.

As another example, if the application program is browsing a blocked, nonkeyed DAM data set and the second logical record from the second physical block on the third relative track is read in response to a GETNEXT request, the record identification field contains

X'00020201'

upon return to the application program, where 0002 represents the track, 02 represents the block, and 01 represents the logical record within the block.

The following examples show how to initiate a browse operation.

For Assembler language:

```
         COPY   DFHTCADS            COPY TCA SYMBOLIC STORAGE DEFN
KEYF     DS     CL8
FWACBAR  EQU    7                   ASSIGN BASE REGISTER FOR FWA
         COPY   DFHFWADS            DEFINE CONTROL SECTION OF FWA
RECORD   DS     0CL350              RECORD LAYOUT
         .
         .
         CSECT
         .
         .
         MVC    KEYF(5),=C'JONES'
         XC     KEYF+5(3),KEYF+5    INITIALIZE KEY FIELD
START    DFHFC  TYPE=SETL,          INITIATE BROWSE                  *
                DATASET=MASTER,                                     *
                RDIDADR=KEYF,                                       *
                NOTOPEN=ERROR       CHECK FOR ERRORS
         L      FWACBAR,TCAFCAA     ESTABLISH ADDRESSABILITY FOR FWA
         .
         .
ERROR    DS     0H                  ENTRY TO ERROR ROUTINE
         .
         .
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                             NOTE DEFINE BASE REGISTER FOR FWA.
      .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PICTURE X(8).           NOTE DEFINE KEY FIELD IN TWA.
01 DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE X(350).       NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.        NOTE ESTABLISH TCA ADDRESSABILITY.
```

```
      .
      .
   MOVE 'JONES' TO KEYF.
START.
        DFHFC TYPE=SETL,           INITIATE BROWSE                  *
              DATASET=MASTER,                                       *
              RDIDADR=KEYF,                                         *
              NOTOPEN=ERROR        CHECK FOR ERRORS
   MOVE TCAFCAA TO FWACBAR.
      .
      .
ERROR.
      .
      .
```

For PL/I:

```
%INCLUDE DFHTCADS;                 /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF CHAR(8);
      .
      .
%INCLUDE DFHFWADS;                 /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);            /*DEFINE RECORD LAYOUT IN FWA*/
      .
      .
KEYF='JONES';
START:
        DFHFC TYPE=SETL,           INITIATE BROWSE                  *
              DATASET=MASTER,                                       *
              RDIDADR=KEYF,                                         *
              NOTOPEN=ERROR        CHECK FOR ERRORS
FWACBAR=TCAFCAA;
      .
      .
ERROR:
      .
      .
```

Retrieve Next Sequential Record (GETNEXT)

Once the application programmer has issued a DFHFC TYPE=SETL macro instruction to initiate a browse operation, the next (or first) sequential record can be obtained by issuing the

```
DFHFC TYPE=GETNEXT,                                                 *
        .
        .
        .
```

macro instruction. When the first GETNEXT request is issued following a SETL request for an ISAM data set, CICS/VS acquires the first logical record with a key equal to or greater than the key presented by a previous SETL; for a DAM data set, CICS/VS acquires the first logical record specified by the user. Each subsequent GETNEXT request, whether for an ISAM or a DAM data set, causes CICS/VS to acquire the next logical record in sequence.

When VSAM is used, the application programmer can specify that a browse operation begin at a particular relative byte location or with a record identified by a key. In the former case, the first GETNEXT request retrieves that record. Each succeeding GETNEXT retrieves the next logical record in sequence.

If a key is specified for a VSAM data set, it may be either specific or generic, and the application programmer can specify that the search begin (1) at a record having a key equal to the specific or generic key, or (2) at a record having a key equal to or greater than the specific or generic key. The effects of GETNEXT macro instructions are as described below.

Before issuing the DFHFC TYPE=GETNEXT macro instruction, the application programmer must place the address of the FWA associated with the particular operation in TCAFCAA. If the application program has initiated multiple browse operations, it must keep track of the FWA associated with each operation and refer to a specific FWA when requiring services related to that browse. Similar requirements apply, but pertain to the address of a specific VSWA when locate-mode processing of VSAM nonsegmented records is utilized.

CICS/VS performs the following services in response to a DFHFC TYPE=GETNEXT macro instruction referring to an ISAM, VSAM, or DAM data set:

1. Retrieves the next sequential record and places it in the FWA specified by the user at TCAFCAA

2. Places the record identification (key, block identification, or the like) of the record just retrieved into the record identification field specified in the DFHFC TYPE=SETL request initiating the browse

If the user issues a DFHFC TYPE=GET,TYPOPER=UPDATE request on the record returned in response to a DFHFC TYPE=GETNEXT request, the address of the record identification field can be specified in the DFHFC TYPE=GET request.

The first DFHFC TYPE=GETNEXT macro instruction referring to a VSAM data set retrieves the record located in response to the DFHFC TYPE=SETL instruction initiating the browse. On a subsequent GETNEXT, CICS/VS checks the contents of the record identification field set aside for records of the data set. If this field contains the identification of the record previously received, CICS/VS retrieves the next logical record in sequence and places the identification of that record in the record identification field. Sequential retrieval such as described above for ISAM and DAM data sets then occurs.

It is possible, however, when using VSAM data sets, for the application programmer to utilize a skip-sequential processing capability. All that is needed is to place the identification of the next record desired into the record identification field prior to issuing a GETNEXT request. If, upon checking this field, CICS/VS determines that its contents have been changed by the application program, CICS/VS accesses the record having the identification currently stored in the record identification field. This record need not be the next sequential record in the data set. This skip-sequential processing capability is available only for VSAM data sets.

When VSAM skip-sequential processing is used, the record identification placed in the record identification field before issuing the GETNEXT request must be of the same form as that specified in the SETL or last RESETL request for this browse operation. That is, if the SETL or last RESETL specified a generic key, then the new record identification must be a generic key. It need not be the same length as that specified in the SETL or last RESETL. If the SETL or last RESETL specified an RBA, the new identification must be an RBA. Note that if the SETL or last RESETL specified an equal search (FKEQ or GKEQ), a GETNEXT request using skip-sequential processing may result in a NOTFND (record not found) condition.

In addition, CICS/VS can perform the following services, depending on the operands included in the DFHFC TYPE=GETNEXT macro instruction.

1. Present the user with segments as specified in the GETNEXT request.

2. Present the user with segments as specified in the SETL request if no segment set is specified in the GETNEXT request.

3. If the FWA is not large enough to process a segment set specified in the GETNEXT request, dispose of the old FWA and acquire a new one large enough to process the new request.

The following examples show how to initiate a browse operation and retrieve selected segments from successive records of the data set. (For more about segmented records, see "Segmented Records" in Chapter 11 and the explanation of the SEGSET operand under "DFHFC Macro Instruction" in Chapter 7.)

For Assembler language:

```
        COPY   DFHTCADS             COPY TCA SYMBOLIC STRG DEFN
KEYF    DS     8X                   DEFINE KEY FIELD IN TWA
FWACBAR EQU    7                    ASSIGN FWA BASE REGISTER
        COPY   DFHFWADS             COPY CICS/VS CONTROL SECTION OF FWA
RECORDA DS     0CL350               DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
        CSECT
        MVC    KEYF(8),=8X'00'      START AT BEGINNING OF DATA SET
INITIAL DFHFC  TYPE=SETL,           INITIATE BROWSE                    *
               DATASET=MASTER,                                         *
               SEGSET=A,            SET DEFAULT SEGMENT SET            *
               RDIDADR=KEYF
        .
        .
        .
        L      FWACBAR,TCAFCAA      ESTABLISH FWA BASE REGISTER
        .
        .
        .
        ST     FWACBAR,TCAFCAA      RESTORE FWA ADDRESS
        DFHFC  TYPE=GETNEXT         GET NEXT SEQUENTIAL RECORD
        L      FWACBAR,TCAFCAA      ASSURE ADDRESSABILITY IF SEGMENTED
        .
        .
        .
        ST     FWACBAR,TCAFCAA      RESTORE FWA ADDRESS
        DFHFC  TYPE=GETNEXT,        GET NEXT RECORD                    *
               SEGSET=B             WITH SEGMENT B
        L      FWACBAR,TCAFCAA      ASSURE ADDRESSABILITY IF SEGMENTED
        .
        .
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                             NOTE DEFINE BASE REGISTER FOR FWA.
      .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PICTURE S9(18) USAGE IS COMPUTATIONAL.
                                     NOTE DEFINE KEY FIELD IN TWA.
```

```
        .
        .
        .
01 DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE X(350).        NOTE DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.         NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
   MOVE 0 TO KEYF.                  NOTE START AT BEGINNING OF DATA SET.
        .
        .
INITIAL.
        DFHFC TYPE=SETL,            INITIATE BROWSE                        *
              DATASET=MASTER,                                              *
              SEGSET=A,             SET DEFAULT SEGMENT SET                *
              RDIDADR=KEYF
   MOVE TCAFCAA TO FWACBAR.         NOTE ESTABLISH FWA ADDRESSABILITY.
        .
        .
        .
   MOVE FWACBAR TO TCAFCAA.
        DFHFC TYPE=GETNEXT          GET NEXT SEQUENTIAL RECORD.
        .
        .
        .
   MOVE FWACBAR TO TCAFCAA.
        DFHFC TYPE=GETNEXT,         GET NEXT RECORD                        *
              SEGSET=B              WITH SEGMENT B
   MOVE TCAFCAA TO FWACBAR.         NOTE POSSIBLE NEW FWA.
        .
        .
```

For PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF BINARY FIXED(8,0);       /*DEFINE KEY FIELD IN TWA*/
        .
        .
        .
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);             /*DEFINE RECORD LAYOUT IN FWA*/
        .
        .
        .
KEYF=0;                             /*START AT BEGINNING OF DATA SET*/
        .
        .
        .
INITIAL:
        DFHFC TYPE=SETL,            INITIATE BROWSE                        *
              DATASET=MASTER,                                              *
              SEGSET=A,             SET DEFAULT SEGMENT SET                *
              RDIDADR=KEYF
FWACBAR=TCAFCAA;                    /*ESTABLISH FWA ADDRESSABILITY*/
        .
        .
        .
TCAFCAA=FWACBAR;
        DFHFC TYPE=GETNEXT          GET NEXT SEQUENTIAL RECORD
```

```
         .
         .
         .
TCAFCAA=FWACBAR;
        DFHFC TYPE=GETNEXT,        GET NEXT RECORD                    *
              SEGSET=B             WITH SEGMENT B
FWACBAR=TCAFCAA;                   /*POSSIBLE NEW FWA*/
         .
         .
```

Terminate Sequential Retrieval (ESETL)

The application programmer can terminate a browse operation by issuing the

```
       DFHFC TYPE=ESETL,                                               *
                .
                .
```

macro instruction. Before the macro is issued, the programmer must ensure that TCAFCAA contains the address of the file work area (FWA) associated with the browse operation he wishes tc terminate. When locate-mode processing of VSAM nonsegmented records is utilized, TCAFCAA must contain the address of the VSAM work area (VSWA) associated with the browse operation being terminated. In response to an ESETL request, CICS/VS releases all I/O and work areas associated with the browse operation.

The following examples show how to terminate two concurrent browse operations.

For Assembler language:

```
         COPY   DFHTCADS            COPY TCA SYMBOLIC STRG DEFN
FWACELL1 DS     A                  CONTAINS ADDR OF FWA USED
*                                  FOR FIRST BROWSE OPERATION
FWACELL2 DS     A                  CONTAINS ADDR OF FWA USED
*                                  FOR SECOND BROWSE OPERATION
FWACBAR  EQU    7                  ASSIGN FWA BASE REGISTER
         COPY   DFHFWADS           DEFINE FWA SYMBOLIC STORAGE DEFN
RECORD   DS     0CL350             DEFINE RECORD
         .
         .
         CSECT
         .
         .
         MVC    TCAFCAA,FWACELL1   MOVE BROWSE 1 FWA ADDR TO TCA
         DFHFC  TYPE=ESETL         ISSUE ESETL MACRO INSTRUCTION
         MVC    TCAFCAA,FCACELL2   MOVE BROWSE 2 FWA ADDR TO TCA
         DFHFC  TYPE=ESETL         ISSUE ESETL MACRO INSTRUCTION
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
         .                          NOTE DEFINE BASE REGISTER FOR FWA.
         .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 FWACELL1 PICTURE S9(8) USAGE IS COMPUTATIONAL.
   02 FWACELL2 PICTURE S9(8) USAGE IS COMPUTATIONAL.
01 DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE X(350).        NOTE DEFINE RECORD LAYOUT IN FWA.
         .
```

```
      .
   MOVE FWACELL1 TO TCAFCAA.        NOTE PREPARE TO END FIRST BROWSE.
       DFHFC TYPE=ESETL             TERMINATE FIRST BROWSE.
      .
      .
   MOVE FWACELL2 TO TCAFCAA.        NOTE PREPARE TO END 2ND BROWSE.
       DFHFC TYPE=ESETL             TERMINATE SECOND BROWSE.
```

For PL/I:

```
%INCLUDE DFHTCADS;                   /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 FWACELL1 POINTER;
   02 FWACELL2 POINTER;
      .
      .
%INCLUDE DFHFWADS;                   /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);              /*DEFINE RECORD LAYOUT IN FWA*/
      .
      .
      .
TCAFCAA=FWACELL1;                    /*MOVE BROWSE1 FWA ADDR TO TCA*/
       DFHFC TYPE=ESETL
TCAFCAA=FWACELL2;                    /*MOVE BROWSE2 FWA ADDR TO TCA*/
       DFHFC TYPE=ESETL
```

Reset Sequential Retrieval (RESETL)

Once a browse operation has been initiated, the application programmer may, at any time prior to issuing an ESETL request for the browse, reset the search argument to some record other than the next sequential record in the data set. The default segment set name and (for a VSAM data set) the type of search argument used in retrieving records can also be reset by issuing the

```
DFHFC TYPE=RESETL,                                                    *
      .
      .
      .
```

macro instruction. Prior to issuing the request, the application programmer should place the address of the appropriate FWA into TCAFCAA and the new record identification in the record identification field specified in the original SETL request.

The use of the RESETL macro instruction allows the application programmer to avoid issuing an ESETL request followed by another SETL request, and causes CICS/VS to use the same I/O and work area. Upon return from the RESETL request, TCAFCAA contains the address of a new FWA which the user can use for the browse operation.

The RESETL request allows the user to "skip" through his data set in a browse operation with minimal overhead. A similar capability is also available to VSAM users through the GETNEXT instruction.

The following examples show how to reset the search argument and the default segment set for a browse operation. (For more about segmented records, see "Segmented Records" in Chapter 11 and the explanation of the SEGSET operand under "DFHFC Macro Instruction" in Chapter 7.)

For Assembler language:

```
         COPY   DFHTCADS                  COPY TCA SYMBOLIC STRG DEFN
KEYF     DS     D                         DEFINE KEY FIELD IN TWA
FWACBAR  EQU    7                         ASSIGN FWA BASE REGISTER
         COPY   DFHFWADS                  COPY FWA DSECT
RECORD1  DS     0CL350                    DEFINE RECORD WITH SEGSET A
         .
         .
         ORG    RECORD1
RECORD2  DS     0CL250                    DEFINE RECORD WITH SEGSET B
         .
         .
         CSECT
         MVC    KEYF(8),=8X'00'           INITIALIZE KEY FIELD
         DFHFC  TYPE=SETL,                ISSUE INITIAL SETL MACRO        *
                DATASET=MASTER,           FOR DATA SET "MASTER"           *
                RDIDADR=KEYF,             INITIAL SEARCH ARG=0            *
                SEGSET=A                  FOR SEGSET=A
         L      FWACBAR,TCAFCAA           ESTABLISH ADDRESSABILITY TO FWA
         .
         .
         ST     FWACBAR,TCAFCAA           STORE FWA ADDR IN TCA
         MVC    KEYF(8),=CL8'SMITH'       ESTABLISH NEW SEARCH ARGUMENT
         DFHFC  TYPE=RESETL,              ISSUE RESETL MACRO              *
                SEGSET=B                  NEW SEGSET ID
         L      FWACBAR,TCAFCAA           ESTABLISH ADDRESSABILITY TO FWA
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
         .
         .                               NOTE DEFINE BASE REGISTER FOR FWA.
         .
01 DFHTCADS COPY DFHTCADS.                NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PICTURE S9(18) USAGE IS COMPUTATIONAL.
                                          NOTE DEFINE KEY FIELD IN TWA.
   02 FILLER REDEFINES KEYF.
      03 KEYC PICTURE X(8).
         .
         .
01 DFHFWADS COPY DFHFWADS.                NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD1 PICTURE X(350).             NOTE DEFINE RECORD WITH SEGSET A.
         .
         .
01 DFHFWA REDEFINES DFHFWADS.             NOTE CREATE STRG DEFN FOR FWA.
   02 CICS/VS/VSPART PICTURE X(*).            NOTE PLACE LENGTH OF FWA HERE.
   02 RECORD2 PICTURE X(250).             NOTE DEFINE RECORD WITH SEGSET B.
         .
         .
   MOVE 0 TO KEYF.
   DFHFC TYPE=SETL,                       ISSUE INITIAL SETL MACRO INSTR  *
              DATASET=MASTER,             FOR DATA SET "MASTER"           *
              RDIDADR=KEYF,               INITIAL SEARCH ARG=0            *
              SEGSET=A                    FOR SEGSET=A
   MOVE TCAFCAA TO FWACBAR.               NOTE ESTABLISH ADDRESSABILITY TO FWA.
         .
         .
   MOVE FWACBAR TO TCAFCAA.               NOTE STORE FWA ADDRESS IN TCA.
   MOVE 'SMITH' TO KEYC.                  NOTE ESTABLISH NEW SEARCH ARGUMENT.
   DFHFC TYPE=RESETL,                     ISSUE RESETL MACRO INSTRUCTION  *
              SEGSET=B                    NEW SEGSET ID
   MOVE TCAFCAA TO FWACBAR.               NOTE ESTABLISH ADDRESSABILITY TO FWA.
```

For PL/I:

```
%INCLUDE DFHTCADS;                    /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF BINARY FIXED(8,0);         /*DEFINE KEY AS BINARY*/
DECLARE 01 DFHXTCA BASED(TCACBAR),
   02 FILL CHAR(*),                   /*PLACE LENGTH OF TCA HERE*/
   02 KEYC CHAR(8);                   /*DEFINE KEY AS CHARACTER*/
      .
      .
      .
%INCLUDE DFHFWADS;                    /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD1 CHAR(350);              /*DEFINE RECORD WITH SEGSET A*/
DECLARE 01 DFHXFWA BASED(FWACBAR),
   02 FILL CHAR(*),                   /*PLACE LENGTH OF FWA HERE*/
   02 RECORD2 CHAR(250);              /*DEFINE RECORD WITH SEGSET B*/
      .
      .
      .
KEYF=0;                               /*SET KEY VALUE TO ZERO*/
        DFHFC TYPE=SETL,              ISSUE INITIAL SETL MACRO INSTR      *
              DATASET=MASTER,         FOR DATA SET "MASTER"               *
              RDIDADR=KEYF,           INITIAL SEARCH ARG EQUALS ZERO      *
              SEGSET=A                FOR SEGSET A
FWACBAR=TCAFCAA;                      /*ESTABLISH ADDRESSABILITY FOR FWA*/
      .
      .
      .
TCAFCAA=FWACBAR;                      /*STORE FWA ADDR IN TCA*/
KEYC='SMITH';                         /*ESTABLISH NEW SEARCH ARGUMENT*/
        DFHFC TYPE=RESETL,            ISSUE RESETL MACRO INSTRUCTION      *
              SEGSET=B                NEW SEGSET ID
FWACBAR=TCAFCAA;                      /*ESTABLISH ADDRESSABILITY TO FWA*/
```

TEST RESPONSE TO A REQUEST FOR FILE SERVICES

When the application programmer issues a request for file services, the CICS/VS response can be used to determine subsequent processing. A preliminary step is to specify the symbolic addresses of user-written exception-handling routines, any of which may be executed as a result of the check. The programmer can do this in any of three ways:

1. Include the symbolic addresses in operands of the DFHFC macro instruction by which the file service is requested.

2. Include the symbolic addresses in operands of a

```
DFHFC TYPE=CHECK,                                                    *
      .
      .
      .
```

   macro instruction immediately following the DFHFC macro instruction by which the service is requested.

3. Include instructions immediately following the DFHFC macro instruction that examine the response code set automatically by CICS/VS when making the response, and transfer control to an appropriate user-written exception-handling routine accordingly.

Under either of the first two methods above, CICS/VS checks the response code that it sets and transfers control to the exception-handling routine named in the operand associated with the condition that has occurred (if that operand has been specified). The application programmer need not be concerned with which response code

corresponds to which condition. It is only necessary to understand the keyword operands and be sure that all conditions that may occur have been provided for.

When the third approach above is used, the application programmer must know (1) the CICS/VS response codes and their meanings, and (2) the symbolic label by which he can refer to the response codes. If the Assembler-language or PL/I programmer elects to check for a particular response-code bit pattern, he can access the response code at TCAFCTR. The American National Standard (ANS) COBOL programmer who elects to check for a particular response-code bit pattern, can access the response code at TCAFCRC. The possible response codes and the conditions to which they correspond are identified in the right-hand columns of Figure 6-1. DFHFC macro instructions for which the conditions are applicable are shown at the left.

Note: Because the multipunch codes to be checked in an ANS COBOL program commonly correspond to unprintable characters, an alternative facility is provided in CICS/VS for use by the ANS COBOL programmer. He can simply refer to the response code by means of a label, formed as a two-character identification of the CICS/VS management module providing the requested service followed by the keyword for the condition being checked (for example, FCNORESP). Use of this approach is illustrated in the examples at the end of this discussion.

If the application programmer does not provide for the checking for a particular response to a file service request, and if the exception condition corresponding to that response occurs, program flow proceeds to the instruction following the file service request in the application program.

The keyword operands that can be used to request tests of the response to a particular request for file services (that is, a particular DFHFC macro instruction) are identified in the discussions of the instruction format and operands under "DFHFC Macro Instruction." The condition expressed by each keyword is explained in detail and should be referred to by the application programmer when using any of the checking methods described above.

Note: When certain exception conditions occur during a read-only operation, an FIOA or VSWA containing the address of the FCT data set entry that produced the exception condition is retained. Its address is available to the application program. Before issuing other file control requests, the application programmer should free the storage occupied by the FIOA or VSWA through use of the DFHFC TYPE=RELEASE macro instruction. (For details see DFHFC TYPE=CHECK in Chapter 7.)

The following examples show how to examine the response code provided by CICS/VS at TCAFCTR (for Assembler language or PL/I) or TCAFCRC (for ANS COBOL) and transfer control to an appropriate user-written error-handling routine. The alternative approach available to ANS COBOL programmers is also exemplified.

File Services Request by DFHFC Macro Instruction	Condition	Response Code		
		Assembler	ANS COBOL	PL/I
ALL	NORESP (Normal Response)	X'00'	12-0-1-8-9 (FCNORESP)	00000000
GET, DELETE, GETAREA, SETL, CHECK	DSIDER (Data Set Identification Error)	X'01'	12-1-9 (FCDSIDER)	00000001
GET, SETL, GETNEXT, RESETL, CHECK	SEGIDER (Segment Set Identification Error)	X'04'	12-4-9 (FCSEGIDER)	00000100
ALL	INVREQ (Invalid Request)	X'08'	12-8-9 (FCINVREQ)	00001000
GET, CHECK	DUPDS (Duplicate Data Set)	X'0A'	12-2-8-9 (FCDUPDS)	00001010
GET, PUT, DELETE, GETAREA, SETL, GETNEXT, RESETL, CHECK	NOTOPEN (Data Set Not Open)	X'0C'	12-4-8-9 (FCNOTOPEN)	00001100
GETNEXT, CHECK	ENDFILE (End of File during Browse)	X'0F'	12-7-8-9 (FCENDFILE)	00001111
GET, DELETE, SETL, GETNEXT, RESETL, CHECK	NOTFND (Record Not Found)	X'81'	12-0-1 (FCNOTFND)	10000001
PUT, CHECK	DUPREC (Duplicate Record)	X'82'	12-0-2 (FCDUPREC)	10000010
PUT, CHECK	NOSPACE (No DASD Space for Adding Record)	X'83'	12-0-3 (FCNOSPACE)	10000011
GET, PUT, DELETE, SETL, GETNEXT, RESETL, CHECK	IOERROR (All Not Covered Above)	X'80'	12-0-1-8 (FCIOERROR)	10000000
GET, PUT, DELETE, SETL, GETNEXT, RESETL, CHECK	ILLOGIC (VSAM Only; Error Not Covered Above)	X'02'	12-2-9 (FCILLOGIC)	00000010
ALL	ERROR (Any Response Other Than NORESP)	See Note 8	See Note 8	See Note 8

NOTES:
1. DELETE is valid only for VSAM files.
2. The SEGIDER condition can occur only when the SEGSET operand is specified.
3. The DUPDS condition can occur only when the INDEX operand is specified.
4. The NOTFND condition can occur for SETL, GETNEXT, or RESETL only when SRCHTYP=FKEQ or SRCHTYP=GKEQ.
5. The NOSPACE condition can occur only when TYPOPER=NEWREC is specified.
6. The ILLOGIC condition can occur only when processing VSAM files.
7. The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.
8. The test for the ERROR response is satisfied by a not equal condition; that is, not X'00', not 12-0-1-8-9, or not 00000000 for Assembler, ANS COBOL, and PL/I, respectively.

Figure 6-1. File Control Response Codes

For Assembler language:

```
       DFHFC TYPE=GET,                                                 *
             DATASET=MASTER,                                           *
             RDIDADR=KEYF
       CLI   TCAFCTR,X'00'          NORMAL RESPONSE
       BE    GOOD
       CLI   TCAFCTR,X'80'          I/O ERROR
       BE    ERROR
       CLI   TCAFCTR,X'08'          INVALID REQUEST
       BE    ERROR
       .
       .
       .
GOOD   DS    0H
       .
       .
       .
ERROR  DS    0H
       DFHPC TYPE=ABEND
```

For ANS COBOL:

```
       DFHFC TYPE=GET,                                                 *
             DATASET=MASTER,                                           *
             RDIDADR=KEYF
       IF TCAFCRC=' ' THEN GO TO GOOD.      NOTE 12-0-1-8-9 NORESP.
       IF TCAFCRC=' ' THEN GO TO ERROR.     NOTE 12-0-1-8  IOERROR.
       IF TCAFCRC=' ' THEN GO TO ERROR.     NOTE 12-8-9    INVREQ.
       .
       .
       .
GOOD.
       .
       .
       .
ERROR.
       DFHPC TYPE=ABEND
```

where the value specified within single quotation marks is an unprintable multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

The alternative approach to response code checking, which is available to ANS COBOL programmers as described earlier, is generally a coding convenience and provides concise code documentation. When this approach is used, the IF statements above are replaced by statements of the form shown below:

```
       IF FCNORESP THEN GO TO GOOD.
       IF FCIOERROR THEN GO TO ERROR.
       IF FCINVREQ THEN GO TO ERROR.
       .
       .
       .
```

For PL/I:

```
       DFHFC TYPE=GET,                                                 *
             DATASET=MASTER,                                           *
             RDIDADR=KEYF
       IF TCAFCTR='00000000'B THEN GO TO GOOD;   /* NORMAL RESPONSE */
       IF TCAFCTR='10000000'B THEN GO TO ERROR;  /* I/O ERROR */
```

```
        IF TCAFCTR='00001000'B THEN GO TO ERROR;  /* INVALID REQUEST */
        .
        .
        .
GOOD:
        .
        .
        .
ERROR:
        DFHPC TYPE=ABEND
```


TRANSIENT DATA SERVICES

Transient data management provides, through transient data control, a generalized queuing facility. Data can be queued (stored) for subsequent internal or external processing. Selected units of information, as specified by the application programmer, can be routed to or from predefined symbolic destinations, either intrapartition or extrapartition. The definitions for the destinations must be contained in a destination control table (DCT) established by the system programmer at system generation.

Intrapartition destinations are queues of data on direct access storage devices developed for input to one or more programs running asynchronously (concurrently) as separate tasks; they are internal to the CICS/VS partition/region. Data directed to or from these internal destinations is called intrapartition data and must consist of variable-length records. Intrapartition destinations can be associated with either a terminal or an output data set. Intrapartition data may be ultimately transmitted upon request to a destination terminal or retrieved sequentially from the output data set. Typical uses of this facility involve message switching, broadcasting, data base access and routing of output to multiple terminals (for example, for order distribution), queuing of data (for example, for assignment of order numbers or priority by arrival), and data collection (for example, for batched input from 2780 Data Transmission Terminals). If generated within the system, the CICS/VS asynchronous transaction processing (ATP) facility can be used to transfer data to or from an intrapartition destination. (See "Asynchronous Transaction Processing" in Chapter 11.)

An intrapartition queue is reusable. The system programmer can indicate, by symbolic destination, whether (1) transient data space management is to control the reuse of tracks associated with a particular destination identification (DESTID), or (2) the releasing of track space is to be controlled through use of the transient data PURGE macro instruction. If transient data space management is not used, an intrapartition queue continues to grow, irrespective of whether the data has been read, until the application programmer purges it.

Extrapartition destinations are queues (data sets) external to the CICS/VS partition/region, residing on any sequential device (DASD, tape, printer, and so on). In general, sequential extrapartition destinations are used for storing data external to the CICS/VS partition/region or for retrieving data from outside the partition/region. For example, one task may read data from a remote terminal, edit the data, and write the results to a data set for subsequent processing in another partition/region. Logging data, statistics, and transaction error messages are examples of data that can be written to extrapartition destinations. In general, extrapartition data created by CICS/VS is intended for subsequent batched input to non-CICS/VS programs. Data can also be routed to an output device such as a line printer. The CICS/VS asynchronous

transaction processing (ATP) facility can be used when reading or writing extrapartition data sets.

Data directed to or from an external destination is called extrapartition data and consists of sequential records that are fixed- or variable-length, blocked or unblocked. The record format for a particular extrapartition destination must be described by the system programmer when setting up the destination control table (see the CICS/VS System Programmer's Reference Manual).

Intrapartition and extrapartition destinations can be used as indirect destinations, which are symbolic references to still other destinations. This facility provides some flexibility in program maintenance in that data can be routed to a destination known by a different symbolic name, without the necessity of recompiling existing programs that use the original name. Only the destination control table need be changed. The application programs can route data to the destination using the previous symbolic name; however, the previous name is now an indirect destination that refers to the new symbolic name. Since indirect destinations are established by means of destination control table entries, the application programmer usually need not be concerned with how this is done. Further information is available in the CICS/VS System Programmer's Reference Manual.

For intrapartition destinations, CICS/VS provides the option of automatic task initiation. A basis for automatic task initiation is established by the system programmer by specifying a nonzero trigger level for a particular intrapartition destination in the DCT. (See discussion of the DFHDCT TYPE=INTRA macro instruction in the CICS/VS System Programmer's Reference Manual.) When the number of entries (PUTs from one or more programs) in the queue (destination) reaches the specified level, a transaction specified in the definition of the destination is automatically initiated. Control is passed to a program that processes the data in the queue; the program must issue repetitive GETs to deplete the queue.

Once the queue has been depleted, a new automatic task initiation cycle begins. That is, a new task is scheduled for initiation when the specified trigger level is again reached, whether or not execution of the prior task has terminated.

If an automatically initiated task does not deplete the queue, access to the queue is not inhibited. The task may be normally or abnormally terminated before the queue is emptied (that is, before a QUEZERO response is returned in response to a DFHTD TYPE=GET macro instruction). If the destination is a terminal, the same task is reinitiated regardless of the trigger level. If the destination is a data set, the task is not reinitiated until the specified trigger level is reached. If the trigger level of a queue is zero, no task is automatically initiated. To ensure that termination of an automatically initiated task occurs when the queue is empty, the application program should test for a QUEZERO condition rather than for some application-dependent factor such as an anticipated number of records. It is the QUEZERO condition only that indicates a depleted queue.

Requests for transient data services are communicated to transient data control through CICS/VS macro instructions. Transient data control then executes as a service program under control of the TCA of the requesting program. It runs at the priority of the requesting program and saves and restores registers from its TCA. After the requested transient data service has been provided (or attempted), control is returned to the next executable instruction in the requesting program.

The transient data management macro instruction (DFHTD) is used to request any of the following services:

1. Direct data to a predefined symbolic destination which references a data set or a terminal

2. Acquire data from a predefined symbolic source which references a data set or a terminal

3. Control the processing of an extrapartition data set

4. Purge data associated with an intrapartition data set

5. Check the response to a request for transient data services

The application programmer must specify the parameters required when requesting transient data services. Parameters can be specified in two ways: (1) by including the parameters in operands of the DFHTD macro instruction by which the service is requested, or (2) by coding instructions that move the required parameters to fields of the TCA prior to issuing the DFHTD macro instruction. The latter approach provides some degree of flexibility in that a single DFHTD macro instruction can be tailored according to current logic needs within the application program.

The application programmer can check the CICS/VS response to a request for transient data services as described under "Test Response to a Request for Transient Data Services." The operands that can be specified in DFHTD macro instructions are explained in detail under "DFHTD Macro Instruction" in Chapter 7.

CICS/VS routes a variety of messages generated by CICS/VS programs or tasks to transient data control. For example, terminal control detects a line or terminal problem (not related to a user-provided task) and routes control to the CICS/VS terminal abnormal condition program (DFHTACP). DFHTACP then generates a message to the control system terminal log (CSTL) and/or to the control system master terminal (CSMT).

Destination definitions for all user and CICS/VS destinations must be included in the destination control table (DCT). Lack of a destination definition leads to an IDERROR (identification error) response to a DFHTD macro instruction.

DISPOSE OF DATA (PUT)

The application programmer can direct transient data to a predefined symbolic destination by issuing the

```
DFHTD TYPE=PUT,                                                    *
        .
        .
        .
```

macro instruction. Destinations are intrapartition if associated with a facility allocated to the CICS/VS partition/region and extrapartition if the data is directed to some destination that is external to the CICS/VS partition/region. If intrapartition data is to be placed in the transient data output area, the symbolic storage definition for this area (DFHTDOA) should be copied in the application program. All references to the output area should be made through the use of a register (TDOABAR) which points to the beginning of the area.

The application programmer must specify the address of the output area containing data to be written, either by specifying the symbolic label of the area in the DFHTD TYPE=PUT macro instruction, or by placing the address in TCATDAA prior to issuing the macro instruction. For

variable-length records and intrapartition data, the first four bytes of the output area must contain the length of the record. The format of this length field is LLØØ, where LL is a two-byte binary length (the value of which includes the length of the data plus the four bytes for the length field) and ØØ should be two bytes containing binary zeros. Transient data ccntrol does not release this area after the data is written as output.

The use of the DFHTD TYPE=PUT macro instructicn is summarized in Figure 6-2.

Macro	Destination Type	Pointer (see note 1)	Output Area	Record Format	Acquired by	Released by (see note 2)
PUT	Extra-partition	First data byte (LLbb if variable-length recs)	Any	Fixed- or variable-length	Application program	DFHSC TYPE=FREEMAIN or task termination
PUT	Intra-partition	LLbb field	TDOA (or any) (see note 3)	Variable-length	Application program	DFHSC TYPE=FREEMAIN or task termination

Notes:
1. TDADDR specified in PUT macro instruction points to the data to be written.
2. DFHSC TYPE=FREEMAIN should be used only if area acquired via DFHSC TYPE=GETMAIN.
3. If the output area is other than TDOA and the system is DOS/VS, an eight-byte storage accounting field must be provided by the application program before the LLbb field. These eight bytes are used by DOS/VS during an intra-partition put and should not be used by the application program.

Figure 6-2. Use of DFHTD TYPE=PUT

The following examples shcw how to write data to a predefined symbolic destination, in this case, the control system message log (CSML). The address of TDOAVRL, the four-byte length field at the beginning of the transient data output area (TDOA), is a pointer to the start of the variable-length data to be written.

For Assembler language:

```
TDOABAR     EQU     7
            COPY    DFHTDOA
DATA        DS      CL10
            .
            .
            MVC     TDOAVRL,LENGTH
```

```
         MVC     DATA,MESSAGE
         MVC     TCATDDI,=C'CSML'
         DFHTD TYPE=PUT,                                          *
               TDADDR=TDOAVRL
         .
         .
```

For ANS COBOL:

```
    01 DFHTDOA COPY DFHTDOA.
        02 DATA X(10).
           .
           .
        02 TDOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
           .
           .
           MOVE LENGTH TO TDOAVRL.
           MOVE MESSAGE TO DATA.
           MOVE 'CSML' TO TCATDDI.
           DFHTD TYPE=PUT,                                        *
                 TDADDR=TDOAVRL
```

For PL/I:

```
%INCLUDE DFHTDOA;
    2  DATA CHAR(10);
           .
           .
           .
       TDOAVRL=LENGTH;
       DATA=MESSAGE;
       TCATDDI='CSML';
       DFHTD TYPE=PUT,                                          *
             TDADDR=TDOAVRL
           .
           .
           .
```

ACQUIRE QUEUED DATA (GET)

The application programmer can acquire transient data from a predefined symbolic source by coding the

```
      DFHTD TYPE=GET,                                           *
            .
            .
            .
```

macro instruction. The address of the data acquired is returned at TCATDAA.

If the data is extrapartition, TCATDAA points to the first word of the data area. For variable-length records, the first four bytes of this area contain the length (LLbb) as specified for variable-length data sets.

If the data is intrapartition, the symbolic storage definition for the transient data input area (DFHTDIA) must have been copied in the application program. TCATDAA points to a CICS/VS input area defined by DFHTDIA. TDIAIRL contains the length (data length plus the length of the length field) of the area.

Transient data (either intrapartition or extrapartition) must be moved from the input area before it can be used in any other input/output operation.

If the application programmer issues a DFHTD TYPE=GET macro instruction, the input area acquired for the previous GET is reused if it is long enough to contain the input record. If it is not, CICS/VS acquires a new input area of sufficient length and releases the input area previously used. If the application programmer issues a DFHTD TYPE=PUT macro instruction, the input area acquired for a previous GET may also be changed or released. The application programmer should always move data to be saved from the input area to a user area to ensure that it is not overlaid with new data. Addressability to the area should also be reestablished following each GET.

The application programmer should not attempt to free storage acquired by the transient data control program in response to a DFHTD TYPE=GET macro instruction. This storage is freed by CICS/VS in the case of intrapartition data, or by the operating system in the case of extrapartition data. An attempt to free storage acquired for extrapartition data may result in an abnormal termination of CICS/VS, since the storage area address returned by transient data control points to storage that is not part of the CICS/VS dynamic storage subpool.

The use of the DFHTD TYPE=GET macro instruction is summarized in Figure 6-3.

Macro	Destination Type	Pointer	Input Area	Record Format	Acquired by	Released by
GET	Extra-partition	TCATDAA	Access-method area	Fixed- or variable-length	Operating system OPEN	Operating system
GET	Intra-partition	TCATDAA	TDIA	Variable-length	CICS/VS	CICS/VS

Note: CICS/VS moves pointer to input data to TCATDAA after GET. For extrapartition, points to LLbb if variable-length. For intra-partition, points to TDIA.

Figure 6-3. Use of DFHTD TYPE=GET

The following examples show how to read a variable-length record from an intrapartition data set specified prior to issuing the DFHTD TYPE=GET macro instruction. In these examples, the data set is the control system message log (CSML).

For Assembler language:

```
TDIABAR    EQU     7
```

```
         COPY    DFHTDIA
         .
         .
         .
         MVC     TCATDDI,=C'CSML'
         DFHTD   TYPE=GET
         L       TDIABAR,TCATDAA
```

For ANS COBOL:

```
         02  TDIABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
         .
         .
     01  DFHTDIA COPY DFHTDIA.
         .
         .
     MOVE 'CSMI' TO TCATDDI.
         DFHTD TYPE=GET
     MOVE TCATDAA TO TDIABAR.
         .
         .
         .
```

For PL/I:

```
     %INCLUDE DFHTDIA;
         2   DUMMY CHAR(1);
         .
         .
         .
     TCATDDI='CSML';
     DFHTD TYPE=GET
     TDIABAR=TCATDAA;
         .
         .
         .
```

Assume that, in the above examples, the variable-length record is read from an extrapartition data set. The address placed at TCATDAA by CICS/VS is the address of the length (LLbb) field that precedes the actual data. Since the DFHTDIA symbolic storage definition is being used, the address must be adjusted to point to the CICS/VS control section preceding the actual data. Therefore, an instruction to adjust the address should be inserted immediately following the instruction that moves the contents of TCATDAA to TDIABAR. The following examples apply to CICS/OS/VS but are applicable to CICS/DOS/VS if '36' is replaced by '8'.

For Assembler language:

```
     SH TDIABAR,=H'36'
         .
         .
```

For ANS COBOL:

```
     SUBTRACT 36 FROM TDIABAR.
         .
         .
```

For PL/I:

```
DCL TDIABAA FIXED BIN(30) BASED(TDIABAB);
TDIABAB=ADDR(TDIABAR);          /* OVERLAY POINTER */
TDIABAA=TDIABAA - 36;           /* DO POINTER ARITHMETIC */
    .
    .
```

Since these examples deal with variable-length records, the first byte of the data is assumed to be the length field (LLbb). If the examples dealt with fixed-length records, appropriate values would be 40 and 12 for CICS/OS/VS and CICS/DOS/VS, respectively.

Note: These values are subject to change in future versions of CICS/VS, because this DSECT is intended only for intrapartition data sets. No DSECT is provided for extrapartition data. Each user should define the extrapartition DSECT so as not to use the absolute values in the above example.

If an extrapartition data set is blocked, alignment requirements are the user's responsibility. CICS/VS assumes that the LLbb field of a variable-length record or the first data byte of a fixed-length record is positioned on a doubleword boundary.

CONTROL THE PROCESSING OF AN EXTRAPARTITION DATA SET (FECV)

The application programmer can create a "forced end of volume" situation on an extrapartition magnetic tape data set by issuing the

```
DFHTD TYPE=FEOV,                                                *
            .
            .
            .
```

macro instruction. This macro instruction is used to cause the rewinding and unloading of a magnetic tape reel. Output labels are created as required and new input labels are verified according to host operating system forced-end-of-volume processing. The next tape reel must then be loaded.

Note: This facility should be used with caution, since CICS/VS operation is halted until the new tape reel has been loaded.

The following examples show how to create a "forced end of volume" situation on an extrapartition magnetic tape data set.

For Assembler language:

```
    .
    .
    .
    MVC   TCATDDI,=C'CSML'
    DFHTD TYPE=FEOV
    .
    .
    .
```

For ANS COBOL:

```
    .
    .
    .
    MOVE 'CSMI' TO TCATDDI.
```

```
      DFHTD TYPE=FEOV
      .
      .
      .
```

For PL/I:

```
      .
      .
      .

      TCATDDI='CSML';
      DFHTD TYPE=FEOV
      .
      .
      .
```

PURGE INTRAPARTITION DATA (PURGE)

When transient data associated with a particular intrapartition destination (queue) is no longer needed, the application programmer can purge the data associated with that destination by issuing the

```
      DFHTD TYPE=PURGE,                                          *
            .
            .
            .
```

macro instruction, which causes all storage associated with the destination to be freed (deallocated).

The DFHTD TYPE=PURGE macro instruction must be used to free storage associated with a destination designated as nonreusable in the destination control table. Otherwise, the storage remains allocated to the destination; the data and amount of storage associated with the destination continue to grow whenever a DFHTD TYPE=PUT macro instruction refers to the destination.

TEST RESPONSE TO A REQUEST FOR TRANSIENT DATA SERVICES

When the application programmer issues a request for transient data services, he can check the CICS/VS response to his request to determine, in a deliberate manner, subsequent processing that should be carried out. One step in doing so is to specify the symbolic addresses of user-written exception-handling routines, any of which may be executed as a result of the check. The available methods for doing so are very similar to those used in checking the response to a request for file services (see "Test Response to a Request for File Services" in this chapter). They are:

- Include the symbolic addresses in operands of the DFHTD macro instruction by which the transient data service is requested.
- Include the symbolic addresses in operands of a

```
      DFHTD TYPE=CHECK,                                          *
            .
            .
            .
```

  macro instruction immediately following the DFHTD macro instruction by which the service is requested.

- Include instructions immediately following the DFHTD macro instruction that test the response code set automatically by CICS/VS, and transfer control to an exception-handling routine accordingly.

The discussion of these methods given for file services is also applicable to transient data services. However, the Assembler-language or PL/I programmer accesses transient data response codes at TCATDTR; the American National Standard (ANS) COBOL programmer accesses these response codes at TCATDRC. In addition, the ANS COBOL programmer can refer to the response codes by means of symbolic labels (TDNORESP, TDQUEZERO, and so on) to cause specific response code patterns to be checked without specifying those patterns in the program. (See the examples at the end of this discussion.) The possible response codes and their meanings are shown in Figure 6-4.

Transient Data Request by DFHTD Macro Instruction	Condition	Response Code		
		Assembler	ANS COBOL	PL/I
ALL	NORESP (Normal Response)	X'00'	12-0-1-8-9 (TDNORESP)	00000000
GET, CHECK	QUEZERO (Queue Is Zero)	X'01'	12-1-9 (TDQUEZERO)	00000001
ALL	IDERROR (Identification Error)	X'02'	12-2-9 (TDIDERROR)	00000010
PUT, GET, CHECK	IOERROR (Input/Output Error)	X'04'	12-4-9 (TDIOERROR)	00000100
PUT, GET, FEOV, CHECK	NOTOPEN (Destination Not Open)	X'08'	12-8-9 (TDNOTOPEN)	00001000
PUT, CHECK	NOSPACE (No Space on Intrapartition Queue, or Write Not Serviceable)	X'10'	12-11-1-8-9 (TDNOSPACE)	00010000

NOTE: The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.

Figure 6-4. Transient Data Control Response Codes

If the application programmer does not check for a particular response to a service request, and if the exception condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program.

The keyword operands that can be used to request tests of the response to a particular request for transient data services (that is, a particular DFHTD macro instruction) are identified in the discussions of the instruction format and operands under "DFHTD Macro Instruction." The condition expressed by each keyword is explained in detail and should be referred to by the application programmer when using any of the checking methods described above.

The following examples show how to examine the response code provided by CICS/VS and transfer control to the appropriate user-written exception-handling routine.

For Assembler language:

```
       DFHTD TYPE=GET,                                              *
             DESTID=CSML
       CLI   TCATDTR,X'00'            NORMAL RESPONSE
       BE    GOOD
       DFHPC TYPE=ABEND
GOOD   DS    0H
       .
       .
       .
```

For ANS COBOL:

```
       DFHTD TYPE=GET,                                              *
             DESTID=CSML
       IF TCATDRC=' ' THEN GO TO GOOD.  NOTE  12-0-1-8-9  NORESP.
       DFHPC TYPE=ABEND
GOOD.
       .
       .
       .
```

where the value specified within single quotation marks is an unprintable multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

The alternative approach to response code checking available to ANS COBOL programmers is generally a coding convenience and provides concise code documentation. When this approach is used, the IF statement above is replaced by a statement of the form shown below:

```
           IF TDNORESP THEN GO TO GOOD.
           .
           .
           .
```

For PL/I:

```
       DFHTD TYPE=GET,                                              *
             DESTID=CSML
       IF TCATDTR='00000000'B THEN GO TO GOOD; /* NORMAL RESPONSE */
       DFHPC TYPE=ABEND
GOOD:
       .
       .
       .
```

TEMPORARY STORAGE SERVICES

Temporary storage management provides the facility, through temporary storage control, that enables user-written application programs to store temporary data in main storage or on auxiliary storage on a direct access storage device. Temporary data is stored, retrieved, and released using a symbolic name (up to eight characters) assigned to the data by the originating task. (The symbolic name cannot consist solely of binary zeros.) The data may be a single unit of information

or information retrieved from or added to a temporary storage message set. The former provides a typical "scratch pad" facility. The latter is designed primarily for terminal paging. It is used in conjunction with basic mapping support (see Chapter 10) and page supervision programs to achieve random access to general-purpose storage files. In general, the paging facility of temporary storage should be used only when multiple records are involved and random access to those records is necessary. This queuing of message sets should not be used for sequential data. transient data management provides facilities for efficient handling of sequential data sets.

Data placed in temporary storage can remain intact beyond the time that the originating task is active in the system. That is, even after the originating task is terminated and its transaction storage released, data placed in temporary storage can be accessed by other tasks through references to the symbolic name under which it was stored. Temporary data remains intact until released by the originating task or by any other task. Prior to release, it can be accessed any number of times.

When temporary data is released, the space that it occupied is reusable. If the data was in main storage, the storage area becomes part of available dynamic storage. If the data was on auxiliary storage, the physical space that the data occupied becomes available and can be reused for other data.

Temporary data can be retrieved by the originating task or by any other task using the symbolic name assigned to it. The name assigned to a single unit of information must be unique. All information moved to or from a temporary storage message set is referred to by a unique name assigned to the message set. Specific entries (logical records) within a message set are referred to by relative position numbers. To avoid conflicts caused by duplicate names, a naming convention should be established and followed by all programmers. For example, the operator identification, terminal identification, or transaction identification could be appended as a prefix or suffix to each programmer-supplied symbolic name.

Temporary data can be stored in either main or auxiliary storage. Generally, main storage should be used if the data is needed for only short periods of time; auxiliary storage should be used if the data must be kept for extended periods of time. Another consideration is that data stored on auxiliary storage is maintained after CICS/VS termination and can be recovered in a subsequent restart. No attempt is made to recover temporary data in main storage. Main storage might be used to pass data from task to task or for unique storage that allows programs to meet the requirement of CICS/VS that they be quasi-reentrant (serially reusable between entry and exit points of the program).

Some uses of the page queuing facility follow:

1. Terminal paging. A task could retrieve a large master record from a direct access data set, format it into several screen images, store the screen images on temporary storage auxiliary storage, and then ask the terminal operator which "page" (screen image) is desired. The application programmer can provide coding (as a generalized routine or unique to a single application) to advance page by page, advance or back up a relative number of pages, and the like.

2. A suspend data set. Assume a data collection task is in progress on a certain terminal. The task reads in one or more units of input and then allows the terminal operator to interrupt the process. If no interruption occurs (some kind of coded input), the task repeats the data collection process.

If the operator interrupts the data collection stream with coded input, the data collection task writes its "incomplete" data to temporary storage and terminates the task. The terminal is now free for entry of a different transaction (perhaps a high-priority inquiry). When the terminal is available to continue the data collection operation, the operator initiates the task in a "resume" mode, causing the task to recall its suspended data from temporary storage and continue as though it had not been interrupted.

3. An application that accepts input data to be written as output on a preprinted form.

The temporary storage management macro instruction (DFHTS) is used to request any of the following services:

- Acquire data from main or auxiliary storage
- Send data to main or auxiliary storage
- Update data in main or auxiliary storage
- Free temporary data in main or auxiliary storage
- Check the response to a request for temporary storage services

The application programmer must specify parameter values when using the DFHTS macro instruction. Parameters can be specified in either of two ways:

- By including the parameters in operands of the DFHTS macro instruction by which temporary storage services are requested, or
- By coding instructions that place the parameter values in fields of the TCA prior to issuing the DFHTS macro instruction

The second of these approaches provides flexibility in that the parameters of a single DFHTS macro instruction can vary to meet the logic needs of the application program.

The application programmer can check the CICS/VS response to a request for temporary storage services as explained under "Test Response to a Request for Temporary Storage Services." If the programmer does not check for a particular response, and if the condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program. All operands that can be included in the DFHTS macro instruction are discussed in detail under "DFHTS Macro Instruction" in Chapter 7.

STORE OR UPDATE TEMPORARY DATA (PUT OR PUTQ)

The application programmer can send temporary data to main or auxiliary storage by issuing either the

```
DFHTS TYPE=PUT,                                                *
      .
      .
      .
```

or

```
DFHTS TYPE=PUTQ,                                                    *
              .
              .
```

macro instruction. PUT causes data to be written to temporary storage as a single unit of information (logical record). PUTQ causes a unit of information to be written to a message set, or queue, in temporary storage. The unit is written in a relative position that is one beyond the last entry written to the message set. Following a PUTQ request, the relative record number is returned to the user in TCATSRN, a two-byte field.

Temporary data may be written from a temporary storage input/output area (TSIOA) or from another main storage area identified by the application programmer. It must have the standard variable-length format, with the data length specified in the first four bytes. These bytes should contain LL00, where LL is a two-byte binary length field (the value of which includes the length of the data plus the four bytes for the length field) and 00 is a two-byte field of binary zeros. The maximum temporary storage record size is based on user-specified data set characteristics. (See "Temporary Storage Data Set" in the CICS/OS/VS Operations Guide or CICS/DOS/VS Operations Guide.)

Existing temporary storage data can be updated by adding the TYPOPER=REPLACE operand to the PUT or PUTQ request. A PUT with TYPOPER=REPLACE causes the current data identified by DATAID to be released and replaced with the data provided. If the DATAID cannot be found, the PUT operates as a normal PUT without TYPOPER=REPLACE. A PUTQ with TYPOPER=REPLACE also requires coding of the ENTRY operand. The specified ENTRY within the message set is released and replaced with the data provided. If the DATAID cannot be found, the PUTQ operates as a normal PUTQ without TYPOPER=RELEASE.

The following examples show how to write a single record of information to temporary storage.

For Assembler language:

```
TSIOABAR   EQU   7
           COPY  DFHTSIOA
DATA       DS    CL10
           .
           .
           .
           MVC   TSIOAVRL,LENGTH
           MVC   DATA,MESSAGE
           DFHTS TYPE=PUT,                                          *
                 DATAID=UNIQNME,                                    *
                 TSDADDR=TSIOAVRL
           .
           .
           .
```

For ANS COBOL:

```
           02 TSIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
       01  DFHTSIOA COPY DFHTSIOA.
           02 DATA PICTURE X(10).
           .
           .
           .
           MOVE LENGTH TO TSIOAVRL.
           MOVE MESSAGE TO DATA.
```

```
      DFHTS TYPE=PUT,                                                  *
            DATAID=UNIQNME,                                            *
            TSDADDR=TSIOAVRL
          .
          .
          .
```

For PL/I:

```
      %INCLUDE DFHTSIOA;
          2 DATA CHAR(10);
          .
          .
          .
      TSIOAVRL=LENGTH;
      DATA=MESSAGE;
      DFHTS TYPE=PUT,                                                  *
            DATAID=UNIQNME,                                            *
            TSADDR=TSIOAVRL
          .
          .
```

RETRIEVE TEMPORARY DATA (GET OR GETQ)

The application programmer can retrieve temporary data from main or auxiliary storage by issuing either the

```
      DFHTS TYPE=GET,                                                  *
              .
              .
              .
```

or

```
      DFHTS TYPE=GETQ,                                                 *
              .
              .
              .
```

macro instruction. The former causes a single unit of information (logical record) to be retrieved. The latter causes an entry previously written to a temporary storage message set, or queue, to be retrieved. The particular record to be retrieved from a queue is identified by means of an ENTRY parameter in the DFHTS TYPE=GETQ macro instruction, which indicates its relative position within a queue. The relative position of an entry is determined by chronological order of creation. Data written to temporary storage by means of a DFHTS TYPE=PUT macro instruction must be retrieved by a DFHTS TYPE=GET macro instruction. Data written by means of DFHTS TYPE=PUTQ must be retrieved by means of DFHTS TYPE=GETQ.

Data retrieved from temporary storage is placed in a storage area identified by the application programmer or in a temporary storage input/output area (TSIOA) acquired by temporary storage control. The application programmer can request in a DFHTS TYPE=GET macro instruction that temporary data be released after it is retrieved. If he does not, the data is retained until it is released by another task or until CICS/VS is terminated.

The following examples show how to read a single record of information from temporary storage. In these examples, the data is moved to the area identified in the TSDADDR operand of the DFHTS TYPE=GET macro instruction by the application programmer. If the

TSDADDR operand is omitted, the data is moved into a storage area obtained by temporary storage control, and the address of the data is returned at TCATSDA to the application program.

For Assembler language:

```
TSIOABAR   EQU     7
           COPY    DFHTSIOA
                .
                .
       DFHTS TYPE=GET,                                          *
             DATAID=UNIQNME
       L TSIOABAR,TCATSDA
       SH TSIOABAR,=H'8'
                .
                .
```

For ANS COBOL:

```
           02  TSIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
                .
                .
       01  DFHTSIOA COPY DFHTSIOA.
                .
                .
       DFHTS TYPE=GET,                                          *
             DATAID=UNIQNME
       MOVE TCATSDA TO TSIOABAR.
       SUBTRACT 8 FROM TSIOABAR.
                .
                .
```

For PL/I:

```
       %INCLUDE DFHTSIOA;
           2 DATA CHAR(10);
                .
                .
       DFHTS TYPE=GET,                                          *
             DATAID=UNIQNME
       DCL TSIOBAA FIXED BIN (30) BASED (TSIOA BAB);
       TSIOABAR=TCATSDA;
       TSIOBAB=ADDR(TSIOABAR);
       TSIOBAA=TSIOBAA-8;
                .
                .
                .
```

FREE TEMPORARY DATA (RELEASE OR PURGE)

The application programmer can free a storage area used for temporary data by issuing either the

```
       DFHTS TYPE=RELEASE,                                      *
                .
                .
                .
```

or

```
DFHTS TYPE=PURGE,                                              *
      .
      .
      .
```

macro instruction. The former causes the main or auxiliary storage area used for a single record of temporary data (created by means of a DFHTS TYPE=PUT macro instruction) to be freed. The latter causes all existing entries in a temporary storage queue (created by means of DFHTS TYPE=PUTQ macro instructions) to be freed. There is no way to release selected records from a temporary storage message set; in particular, a DFHTS TYPE=RELEASE macro instruction cannot be used to release a record that is part of a message set created by means of DFHTS TYPE=PUTQ.

If temporary data named in a DFHTS TYPE=RELEASE or DFHTS TYPE=PURGE macro instruction was in main storage, the area that it occupied is freed and returned to the available dynamic storage area. If the data was in auxiliary storage, the space is made available for reuse.

Temporary data should be released at the earliest possible time to avoid using excessive amounts of storage for this purpose.

The following examples show how to release a single record from temporary storage.

For Assembler language:

```
MVC   TCATSDI,=C'UNIQNME'
DFHTS TYPE=RELEASE
```

For ANS COBOL:

```
MOVE 'UNIQNME' TO TCATSDI.
DFHTS TYPE=RELEASE
```

For PL/I:

```
TCATSDI='UNIQNME';
DFHTS TYPE=RELEASE
```

TEST RESPONSE TO A REQUEST FOR TEMPORARY STORAGE SERVICES

After issuing a request for for temporary storage services, the application programmer can check the CICS/VS response to the request to determine subsequent processing that should be carried out. One step in performing this check is to specify the symbolic addresses of user-written exception-handling routines, any of which may be executed as a result of the check. This can be accomplished in any of three ways:

1. Include the symbolic addresses in operands of the DFHTS macro instruction by which the temporary storage service is requested.

2. Include the symbolic addresses in operands of a

```
DFHTS TYPE=CHECK,                                              *
      .
      .
      .
```

macro instruction immediately following the DFHTS macro instruction by which the service is requested.

3. Include instructions immediately following the DFHTS macro instruction that examine the response code set automatically by CICS/VS and transfer control to an exception-handling routine accordingly.

The general discussion under "Test Response to a Request for File Services" applies to temporary storage services as well. The Assembler-language or PL/I programmer can access temporary storage response codes at TCATSTR; the ANS COBOL programmer can access temporary storage response codes at TCATSRC. In addition, the ANS COBOL programmer can refer to the response codes by means of symbolic labels (TSNORESP, TSIDERROR, and so on) to cause specific response code patterns to be checked without specifying those patterns in his program. (See the examples at the end of this discussion.) The possible response codes and the conditions to which they correspond are identified in the right-hand columns of Figure 6-5. DFHTS macro instructions for which the conditions are applicable are shown at the left.

Temporary Storage Request by DFHTS Macro Instruction	Condition	Response Code		
		Assembler	ANS COBOL	PL/I
ALL	NORESP (Normal Response)	X'00'	12-0-1-8-9 (TSNORESP)	00000000
GET, GETQ, RELEASE, PURGE, CHECK	IDERROR (Identification Error)	X'02'	12-2-9 (TSIDERROR)	00000010
PUT, PUTQ, GET, GETQ, CHECK	IOERROR (Input/Output Error)	X'04'	12-4-9 (TSIOERROR)	00000100
All	INVREQ (Invalid Request)	X'20'	11-0-1-8-9 (TSINVREQ)	00100000
PUTQ, GETQ, CHECK	ENERROR (Entry Error)	X'01'	12-1-9 (TSENERROR)	00000001
PUT, PUTQ	NOSPACE (No Space on Auxiliary Storage)	X'08'	12-8-9	00001000
All	ERROR (Any of Above but Unspecified)	See Note 2	See Note 2	See Note 2

NOTES:
1. The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.
2. The test for the ERROR response is satisfied by a not equal condition; that is, not X'00', not 12-0-1-8-9, or not 00000000 for Assembler, ANS COBOL, and PL/I, respectively.

Figure 6-5. Temporary Storage Control Response Codes

If the application programmer does not check for a particular response to a service request, and if the exception condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program.

The following examples show how to examine the response code provided by CICS/VS and transfer control to the appropriate user-written exception-handling routine.

For Assembler language:

```
        DFHTS TYPE=GET,                                              *
              DATAID=UNIQNME,                                        *
              TSDADDR=YES
        CLI   TCATSTR,X'00'                NORMAL RESPONSE
        BE    GOOD
        DFHPC TYPE=ABEND
GOOD    DS    0H
        .
        .
        .
```

For ANS COBOL:

```
        DFHTS TYPE=GET,                                              *
              DATAID=UNIQNME,                                        *
              TSDADDR=YES
        IF TCATSRC=' ' THEN GO TO GOOD.    NOTE  12-0-1-8-9 NORESP.
        DFHPC TYPE=ABEND
GOOD.
        .
        .
        .
```

where the value specified within single quotation marks is an unprintable multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

The alternative approach to response code checking available to ANS COBOL programmers is generally a coding convenience and provides concise code documentation. When this approach is used, the IF statement above is replaced by a statement of the form shown below:

```
        IF TSNORESP THEN GO TC GOCD.
        .
        .
        .
```

For PL/I:

```
        DFHTS TYPE=GET,                                              *
              DATAID=UNIQNME,                                        *
              TSDADDR=YES
        IF TCATSTR='00000000'B THEN GO TO GOOD;  /* NORMAL RESPONSE */
        DFHPC TYPE=ABEND
GOOD:
        .
        .
        .
```

STORAGE SERVICES

Storage management controls all dynamic main storage for CICS/VS and for user-written application programs. Requests to acquire or release main storage are communicated to CICS/VS storage control by means of the storage management macro instruction (DFHSC).

CICS/VS management programs issue requests for main storage to provide input/output areas, program load areas, and user-defined work areas needed to process a transaction. An application program can issue requests for main storage to provide intermediate work areas and any other main storage not automatically provided by CICS/VS but needed to process a transaction. Any main storage acquired by an application program can be initialized to any bit configuration the user desires. For example, binary zeros or EBCDIC blanks are a common choice.

All main storage associated with a transaction is controlled and accounted for by CICS/VS. This allows CICS/VS to release all main storage associated with a transaction upon request or when the transaction is normally or abnormally terminated. Main storage is accounted for as follows:

- Task control areas (TCAs) are accounted for through pointers in the dispatch control areas (DCA). The DCAs are chained from the common system area (CSA).
- Transaction storage is chained off the task control area (TCA).
- Terminal storage is chained off the TCTTE (the TCTTESC field is the origin of the terminal input/output area (TIOA) chain; the TCTTEDA field contains the address of the current TIOA regardless of the position of that TIOA on the chain).
- Program storage is accounted for in the program processing table (PPT).
- Suspended tasks are accounted for by the suspending CICS/VS management program (task control, storage control, or temporary storage control).

If there is insufficient main storage to satisfy a storage acquisition request, TCASCSA is filled with binary zeros. All activity within the task is suspended until sufficient dynamic storage becomes available and its address is placed in TCASCSA, unless the application programmer has specified in his request that control is to be returned to the application program. No storage will cause a short-on-storage condition. The initiation of new tasks is restricted by CICS/VS until the short-on-storage condition is alleviated. Normally, this occurs as a result of some other task releasing storage currently reserved for it. (See "Purge a Task on System Overload (PURGE/NOPURGE)" for corrective action that can be taken if the short-on-storage condition continues.)

OBTAIN AND INITIALIZE MAIN STORAGE (GETMAIN)

Requests for main storage are made by issuing the

```
DFHSC TYPE=GETMAIN,                                          *
      .
      .
      .
```

macro instruction. This instruction is used by the application programmer to obtain main storage of a specified size and class and,

optionally, to initialize that storage to a bit configuration the application programmer desires. The address of the storage area obtained upon execution of this instruction is placed in TCASCSA by CICS/VS; the obtained storage is doubleword-aligned.

When using the DFHSC TYPE=GETMAIN macro instruction, the application programmer should:

- Specify the class of storage desired in the DFHSC TYPE=GETMAIN macro instruction.
- Calculate the number of bytes required and either specify that amount in the DFHSC macro instruction, or place it in TCASCNB before issuing the DFHSC macro instruction. A zero data length is not allowed for a DFHSC TYPE=GETMAIN macro instruction.
- Specify a symbolic base address for the storage area.
- Move the storage address located at TCASCSA to the symbolic base address. (This address points to the storage accounting area of the storage area.)
- Copy the symbolic storage definition for the appropriate input/output area or storage accounting area prior to the symbolic definition of the user's program storage area.

The following example shows how to request a 1024-byte area of main storage:

```
DFHSC TYPE=GETMAIN,       OBTAIN NEW STORAGE AREA              *
      INITIMG=00,         INITIALIZE WITH BINARY ZEROS         *
      NUMBYTE=1024,       SIZE OF STORAGE REQUESTED            *
      CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
```

The following examples show how to specify the size of a required storage area and the value to which it is to be initialized and then request that the storage be acquired.

For Assembler language:

```
      MVI   TCASCIB,B'0'        INITIALIZE WITH BINARY ZEROS
      MVC   TCASCNB,=H'1024'    SIZE OF STORAGE REQUESTED
            .
            .
      DFHSC TYPE=GETMAIN,       OBTAIN NEW STORAGE AREA              *
            INITIMG=YES,        INITIALIZE WITH BINARY ZEROS         *
            COND=YES,           RETURN CONTROL IF NO STORAGE         *
            CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
      CLC   TCASCSA,=F'0'       WAS STORAGE AVAILABLE?
      BE    NOSTRG              BRANCH IF NOT
      L     TIOABAR,TCASCSA     LOAD REGISTER IF STORAGE FOUND
```

For ANS COBOL:

```
      MOVE ' ' TO TCACSIB.      NOTE INITIALIZE WITH BLANKS
      MOVE 1024 TO TCASCNB.     NOTE SIZE OF STORAGE REQUESTED.
            .
            .
      DFHSC TYPE=GETMAIN,       OBTAIN NEW STORAGE AREA              *
            INITIMG=YES,        INITIALIZE WITH BLANKS               *
            COND=YES,           RETURN CONTROL IF NO STORAGE         *
            CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
```

```
       IF TCASCSA EQUAL 0 GO TO NOSTRG.
       MOVE TCASCSA TO TIOABAR.
```

For PL/I:

```
       TCASCIB=0;                 /*INITIALIZE WITH BINARY ZEROS*/
       TCASCNB=1024;              /*SIZE OF STORAGE REQUESTED*/
            .
            .
       DFHSC TYPE=GETMAIN,        OBTAIN NEW STORAGE AREA              *
             INITIMG=YES,         INITIALIZE WITH BINARY ZEROS         *
             COND=(NO,NOSTRG),    RETURN CONTROL IF NO STORAGE         *
             CLASS=TERMINAL       CLASS OF STORAGE REQUESTED
       IF TCASCSA = 0 THEN GO TO NOSTRG;
       TIOABAR=TCASCSA;           /*LOAD REGISTER IF STORAGE FOUND*/
```

RELEASE MAIN STORAGE (FREEMAIN)

Previously acquired main storage is released by issuing the

```
       DFHSC TYPE=FREEMAIN,                                            *
                .
                .
```

macro instruction. If the task itself does not release acquired storage, the storage is released by CICS/VS upon termination of the task.

If using the DFHSC TYPE=FREEMAIN macro instruction to release a single storage area, the application programmer must place the address of that area in TCASCSA prior to execution of the DFHSC TYPE=FREEMAIN macro instruction. If all terminal storage acquired by means of DFHSC TYPE=GETMAIN,CLASS=TERMINAL macro instructions in the application program or by CICS/VS on behalf of the task is to be released, the RELEASE=ALL operand can be coded in the DFHSC TYPE=FREEMAIN macro instruction to achieve that result; in this case, it is not necessary to place an address in TCASCSA.

The following example shows how to release all main storage currently allocated to a specific terminal:

```
       DFHSC TYPE=FREEMAIN,                                            *
             RELEASE=ALL          RELEASE ALL TERMINAL STORAGE
```

The following examples show how to release a single main storage area, placing the address of the area to be released in TCASCSA before issuing the release request.

For Assembler language:

```
       ST    TIOABAR,TCASCSA      PLACE STORAGE AREA ADDRESS IN TCA
             .
             .
             .
       DFHSC TYPE=FREEMAIN        RELEASE STORAGE AREA
```

For ANS COBOL:

```
       MOVE TIOABAR TO TCASCSA.   NOTE PLACE STRG AREA ADDR IN TCA.
             .
             .
```

```
      DFHSC TYPE=FREEMAIN          RELEASE STORAGE AREA
```

For PL/I:

```
      TCASCSA=TIOABAR;             /*PLACE STORAGE AREA ADDRESS IN TCA*/
            .
            .
      DFHSC TYPE=FREEMAIN          RELEASE STORAGE AREA
```

PROGRAM SERVICES

All program communication within CICS/VS is accomplished by program management. The program management macro instruction (DFHPC) is used to request any of the following services:

- Link one user-written application program to another, anticipating subsequent return to the requesting program.
- Transfer control from one user-written application program to another, anticipating no return to the requesting program.
- Load a designated application program, table, or map (generally, for use with basic mapping support) into main storage and return control to the requesting program.
- Return control from one user-written application program to another or to CICS/VS.
- Delete a previously loaded application program from main storage.
- Abnormally terminate a transaction and its related task.
- Activate, cancel, or reactivate an exit that permits user-written abnormal termination processing.
- Convert a symbolic label in an ANS COBOL application program into an address (returned in TCAPCLA).

PROGRAM

Application programs running under CICS/VS are executed at various logical levels. For example, where one user-written application program is linked to another, the linked-to program is considered to reside at the next lower logical level. Where control is simply transferred from one application program to another, the two programs are considered to reside at the same logical level. A program control DFHPC TYPE=LINK macro instruction is used for the former; a DFHPC TYPE=XCTL macro instruction (where XCTL means transfer control) is used for the latter. Figure 6-6 illustrates this difference between program linkage and transfer of program control. Each of the programs shown in this figure may have been written in any of the CICS/VS-supported languages (Assembler language, ANS COBOL, or PL/I). Use of LINK, XCTL, RETURN, and ABEND is explained in greater detail below.

Transactions can share the use of common work areas. However, each transaction requires the use of a unique intermediate storage area, such as the transaction work area (TWA), to retain information needed upon subsequent return to that transaction. The application programmer must provide addressability to that intermediate storage area by symbolically defining it in his program.

Parameters can be passed from one program to another through user-defined storage areas, for example, the transaction work area (TWA), the terminal input/output area (TIOA), the terminal control Table terminal entry (TCTTE), or the file work area (FWA).

CICS/VS automatically saves program control information and general-purpose registers, when applicable, in the task control area (TCA). CICS/VS automatically restores general-purpose registers, as necessary, to return control to a program. The name of any program referenced in a request for program services must have been placed in the processing program table (PPT) prior to execution of CICS/VS.

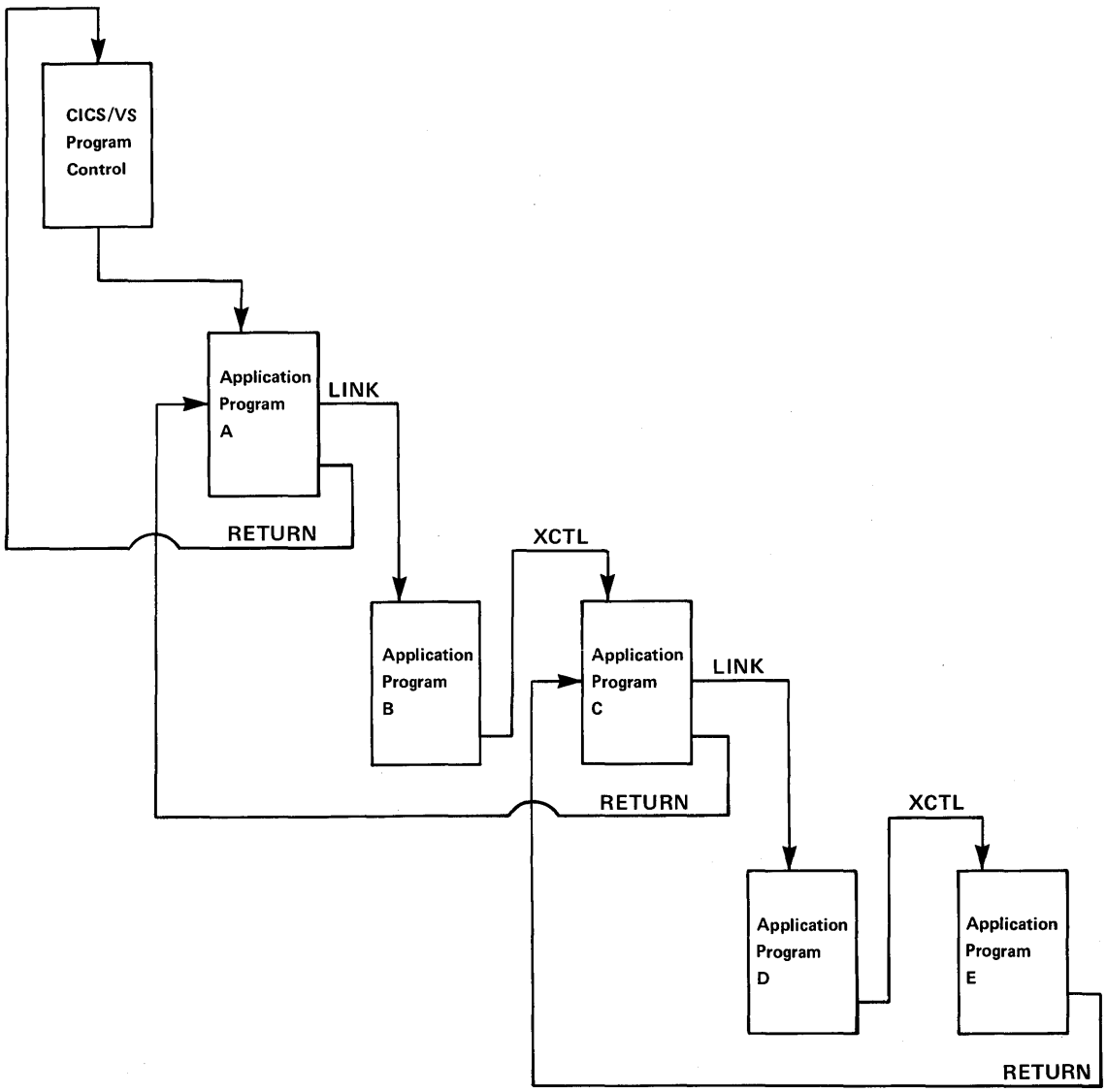


Figure 6-6. Communication and Logical Relationships among Application Programs

PASS PROGRAM CONTROL ANTICIPATING RETURN (LINK)

Program control is passed from a user-written application program at one logical level to a user-written application program at the next lower logical level in response to the

```
DFHPC TYPE=LINK,                                                    *
      .
      .
```

macro instruction. When the DFHPC TYPE=RETURN macro instruction is executed in the linked-to program, control is returned to the program initiating the linkage at the next sequential (executable) instruction.

The application programmer must specify the name (identification) of the program to which control is passed in the DFHPC TYPE=LINK macro instruction or in a single instruction that places the program name in TCAPCPI prior to issuing the DFHPC TYPE=LINK macro instruction.

The following example shows how to request a link to a particular application program:

```
DFHPC TYPE=LINK,                                                    *
      PROGRAM=PROG1
```

The following examples show how to link to an application program specified dynamically by an instruction executed prior to DFHPC TYPE=LINK.

For Assembler language:

```
      MVC   TCAPCPI,=CL8'PROG1'  PLACE LINKED-TO PROGRAM NAME IN TCA
            .
            .
            .
      DFHPC TYPE=LINK            LINK TO PROGRAM AT NEXT LOWER LEVEL
```

For ANS COBOL:

```
      MOVE 'PROG1' TO TCAPCPI.   NOTE LINKED-TO PROGRAM NAME TO TCA.
            .
            .
            .
      DFHPC TYPE=LINK            LINK TO PROGRAM AT NEXT LOWER LEVEL
```

For PL/I:

```
      TCAPCPI='PROG1';           /*PLACE LINKED-TO PRGM NAME IN TCA*/
            .
            .
            .
      DFHPC TYPE=LINK            LINK TO PROGRAM AT NEXT LOWER LEVEL
```

TRANSFER PROGRAM CONTROL (XCTL)

Program control is transferred from one user-written application program to another at the same logical level in response to the

```
DFHPC TYPE=XCTL,                                                    *
      .
      .
```

macro instruction. The program from which control is transferred is released. Any return from the transferred-to program is to a program from which there was an exit at the next higher logical level. If there is no user-written application program at the next higher logical level, control is returned to CICS/VS.

The application programmer must specify the name (identification) of the program to which control is transferred in the DFHPC TYPE=XCTL macro instruction or in a single instruction that places the program name in TCAPCPI prior to issuing the DFHPC TYPE=XCTL macro instruction.

The following example shows how to request a transfer of control to a particular application program:

```
DFHPC TYPE=XCTL,                                              *
      PROGRAM=PROG2
```

The following examples show how to transfer control to an application program specified dynamically by an instruction executed prior to DFHPC TYPE=XCTL.

For Assembler language:

```
MVC   TCAPCPI,=CL8'PRCG2'  PLACE TRANSFERRED-TO PRGM NAME IN TCA
      .
      .
      .
DFHPC TYPE=XCTL            TRANSFER PROGRAM CONTROL
```

For ANS COBOL:

```
MOVE 'PROG2' TO TCAPCPI.   NOTE TRANSFERRED-TO PRGM NAME TO TCA.
      .
      .
      .
DFHPC TYPE=XCTL            TRANSFER PROGRAM CONTROL
```

For PL/I:

```
TCAPCPI='PROG2';           /*PLACE PROGRAM NAME IN TCA*/
      .
      .
      .
DFHPC TYPE=XCTL            TRANSFER PROGRAM CONTROL
```

LOAD A PROGRAM (LOAD)

Programs, tables, or maps are fetched from the library where they reside and loaded into main storage in response to the

```
DFHPC TYPE=LOAD,                                              *
      .
      .
```

macro instruction, which identifies the program to be loaded. This facility is used to (1) load a program that will be used repeatedly, thereby reducing system overhead through a one-time load, (2) load a table to which control is not to be passed, or (3) load a map to be used in a mapping operation (see Chapter 10). CICS/VS returns the address of the loaded program in TCAPCLA.

The loaded program remains in main storage until the DFHPC TYPE=DELETE macro instruction is issued or until the task that issued the DFHPC TYPE=LOAD is terminated, either normally or abnormally (unless LOADLST=NO was specified). If the LOADLST=NO operand is used, the loaded program remains resident until it is deleted by the user.

The application programmer must provide the name (identification) of the program to be loaded in the DFHPC TYPE=LOAD macro instruction or in a single instruction that places the program name in TCAPCPI prior to issuing the DFHPC TYPE=LOAD macro instruction.

The following example shows how to load a particular user-written application program:

```
DFHPC TYPE=LOAD,                                                    *
      PROGRAM=PROG3
```

The following examples show how to load an application program specified dynamically by an instruction executed prior to DFHPC TYPE=LOAD.

For Assembler language:

```
MVC   TCAPCPI,=CL8'PROG3'  PLACE PROGRAM NAME IN TCA
      .
      .
      .
DFHPC TYPE=LOAD            LOAD THE SPECIFIED PROGRAM
```

For ANS COBOL:

```
MOVE 'PROG3' TO TCAPCPI.   NOTE PLACE PRGM NAME IN TCA.
      .
      .
      .
DFHPC TYPE=LOAD            LOAD THE SPECIFIED PROGRAM
```

For PL/I:

```
TCAPCPI='PROG3';           /*PLACE PROGRAM NAME IN TCA*/
      .
      .
      .
DFHPC TYPE=LOAD            LOAD THE SPECIFIED PROGRAM
```

RETURN PROGRAM CONTROL (RETURN)

Program control is returned from an application program to a program at the next higher logical level in response to the

```
DFHPC TYPE=RETURN,                                                  *
      .
      .
```

macro instruction. When this macro instruction is executed in a lower level (linked-to) program, it restores the registers of the higher level (linked-from) program to their contents at the time the DFHFC TYPE=LINK was issued and releases save areas for the lower-level program. In general, the program to which control is returned must have relinquished control by execution of a DFHPC TYPE=LINK macro instruction and must reside one logical level higher than the program returning control. Upon normal termination of transaction processing, control is returned to CICS/VS.

If no default transaction code has been assembled into the terminal control table terminal entry (TCTTE) for a particular terminal, the application programmer can specify the transaction identification for

the next program to be associated with that terminal in either of two ways: (1) by including the desired transaction identification in the DFHPC TYPE=RETURN macro instruction, or (2) by coding a single instruction that places the desired transaction identification in TCANXTID prior to issuing the DFHPC TYPE=RETURN macro instruction. By doing so, the programmer ensures that subsequent unsolicited input can be entered from the terminal without the specification of a transaction identification. A flexible means of starting the next transaction (task) is provided. Any higher-level program specification of transaction identification overrides a specification by a lower-level program.

DELETE A LOADED PROGRAM (DELETE)

A program previously loaded through use of the DFHPC TYPE=LOAD macro instruction with or without the LOADLST=NO operand is deleted (released) by the

```
DFHPC TYPE=DELETE,                                              *
        .
        .
```

macro instruction. If the DFHPC TYPE=LOAD macro instruction contained LOADLST=NO, the loaded program is deleted only in response to a DFHPC TYPE=DELETE macro instruction. If LOADLST=NO was not specified, the loaded program can be deleted by a DFHPC TYPE=DELETE request, or it will be automatically deleted when the task that issued the load request is terminated.

The application programmer must specify the name (identification) of the program to be deleted in the DFHPC TYPE=DELETE macro instruction or in an instruction that places the program name in TCAPCPI prior to issuing the DFHPC TYPE=DELETE macro instruction.

The following example shows how to delete a user-written application program loaded in response to a DFHPC TYPE=LOAD macro instruction.

```
DFHPC TYPE=DELETE,                                              *
      PROGRAM=PROG4
```

The following examples show how to dynamically delete an application program loaded in response to a DFHPC TYPE=LOAD macro instruction.

For Assembler language:

```
MVC   TCAPCPI,=CL8'PROG4'  PLACE PROGRAM NAME IN TCA
        .
        .
        .
DFHPC TYPE=DELETE          DELETE THE SPECIFIED PROGRAM
```

For ANS COBOL:

```
MOVE 'PROG4' TO TCAPCPI.   NOTE PLACE PRGM NAME IN TCA.
        .
        .
        .
DFHPC TYPE=DELETE          DELETE THE SPECIFIED PROGRAM
```

For PL/I:

```
TCAPCPI='PROG4';            /*PLACE PROGRAM NAME IN TCA*/
      .
      .
      .
DFHPC TYPE=DELETE           DELETE THE SPECIFIED PROGRAM
```

ABNORMALLY TERMINATE A TRANSACTION (ABEND)

The application programmer can cause a transaction and its related task to be terminated abnormally by issuing the

```
DFHPC TYPE=ABEND,                                              *
        .
        .
```

macro instruction. If a task is attached by another task, only the task that issues the ABEND is terminated. The main storage associated with the terminated transaction is released. If CANCEL=YES is specified, all exits established by DFHPC TYPE=SETXIT macro instructions at any level in the task are canceled.

The application programmer can request a dump of main storage related to the terminated transaction. The request must specify a four-character abnormal termination code that dump control will place in the formatted storage dump to identify the ABEND condition. This code can be specified in two ways.

1. It can be specified in the TYPE=ABEND macro instruction, as follows.

   ```
   DFHPC TYPE=ABEND,                                           *
         ABCODE=1234
   ```

2. It can be placed in TCAPCAC before issuing the macro instruction, as follows.

For Assembler language:

```
MVC   TCAPCAC,=CL4'1234'    PLACE TERMINATION CODE IN TCA
      .
      .
      .
DFHPC TYPE=ABEND,           TERMINATE PGRM, TRANS, & TASK      *
      ABCODE=YES            USE ABCODE ALREADY SPECIFIED
```

For ANS COBOL:

```
MOVE '1234' TO TCAPCAC.     NOTE TERMINATION CODE TO TCA.
      .
      .
      .
DFHPC TYPE=ABEND,           TERMINATE PGRM, TRANS, & TASK      *
      ABCODE=YES            USE ABCODE ALREADY SPECIFIED
```

For PL/I:

```
TCAPCAC='1234';             /*PLACE TERMINATION CODE IN TCA*/
      .
      .
```

```
          .
DFHPC TYPE=ABEND,          TERMINATE PGRM, TRANS, & TASK      *
      ABCODE=YES           USE ABCODE ALREADY SPECIFIED
```

ACTIVATE, CANCEL, OR REACTIVATE AN EXIT FOR ABNORMAL TERMINATION PROCESSING (SETXIT OR RESETXIT)

During abnormal termination of a task, a program-level ABEND exit is provided in CICS/VS program control so that a user-written exit routine can be executed if desired. One example of a function performed by such a routine is the "clean-up" of a program that has started but not completed normally. The exit is activated in response to the

```
DFHPC TYPE=SETXIT,
          .                                                    *
          .
          .
```

macro instruction. If an abnormal termination condition occurs while this macro instruction is in effect, program control transfers control to the user's exit routine.

The application programmer must specify the name of the program or the address of the routine to be given control in the DFHPC TYPE=SETXIT macro instruction or in an instruction that places the program name in TCAPCPI or the routine address in TCAPCERA prior to issuing the DFHPC TYPE=SETXIT macro instruction.

Only one exit to a user-written exit routine can be active at a time. A DFHPC TYPE=SETXIT macro instruction in which a program or routine name is specified overrides (effectively, replaces) any preceding DFHPC TYPE=SETXIT macro instruction. To simply cancel a previously established exit, the application programmer can issue a DFHPC TYPE=SETXIT macro instruction in which neither the program name nor the routine name operand is specified.

To prevent recursive ABENDs in an exit routine, CICS/VS deactivates an exit upon entry to the exit routine. If attempting a retry of the operation, the programmer can branch to a point in the program that was in control at the time of the ABEND and issue the

```
DFHPC TYPE=RESETXIT
```

macro instruction to reactivate the exit. The user can also use this macro instruction to reactivate an exit that was canceled previously as described above. No additional parameters are required.

Upon entry to an exit program, no addressability can be assumed other than that normally assumed for an application program coded in the language. If the exit logic is in the form of a routine, the amount of addressability varies with the source language, as detailed under "Creating a Task ABEND Exit" in the CICS/VS System Programmer's Reference Manual. For additional information concerning preparation of the exit routine, see that manual.

The following example shows how to establish a program as an exit:

```
DFHPC   TYPE=SETXIT,                                           *
        PROGRAM=EXITPGM
```

The following examples show how to establish a routine as an exit by dynamically storing the address of the routine prior to executing the DFHPC TYPE=SETXIT macro instruction.

For Assembler language:

```
        LA     14,EXITRTN
        ST     14,TCAPCERA
         .
         .
         .
        DFHSC TYPE=SETXIT,                                          *
              ROUTINE=YES
```

For ANS COBOL:

```
        DFHPC TYPE=COBADDR,                                         *
              LABEL=EXITRTN
        MOVE  TCAPCLA TO TCAPCERA
         .
         .
         .
        DFHPC TYPE=SETXIT,                                          *
              ROUTINE=YES
```

For PL/I:

```
        TCAPCERA=ADDR(EXITRTN);
         .
         .
         .
        DFHPC TYPE=SETXIT,                                          *
              ROUTINE=YES
```

CONVERT SYMBOLIC LABEL TO ADDRESS (COBADDR)

The ANS COBOL programmer can request that program control convert a symbolic label to an address by issuing the

```
DFHPC TYPE=COBADDR,
        .
        .
```

macro instruction. The address is returned in TCAPCLA by program control.

A comparable facility is available within both PL/I and Assembler language; this macro instruction is designed to provide the capability for ANS COBOL programmers. ANS COBOL support must have been generated within CICS/VS to support ANS COBOL programs.

TEST RESPONSE TO A REQUEST FOR PROGRAM SERVICES

When the application programmer issues a DFHPC TYPE=LINK, TYPE=LOAD, or TYPE=SETXIT request, the CICS/VS response can be checked to determine subsequent processing that should be carried out. One step in doing so is to specify the symbolic addresses of user-written exception-handling routines, any of which may be executed as a result of the check. The addresses can be specified in three ways:

1. Include the symbolic addresses in operands of the DFHPC macro instruction by which the program service is requested.

2. Include the symbclic addresses in operands of a

```
DFHPC TYPE=CHECK,                                              *
          .
          .
          .
```

macro instruction immediately following the DFHPC macro instruction by which the service is requested.

3. Include instructions immediately following the DFHPC macro instruction which examine the response code set automatically by CICS/VS and transfer control to an exception-handling routine accordingly.

Under either of the first two methods above, CICS/VS checks the response code that is set and transfers control to the exception-handling routine named in the operand associated with the condition that has occurred (if that operand has been specified). The application programmer need not be concerned with which response code corresponds to which condition. It is only necessary to understand the keyword operands and be sure that he has provided for all conditions that may occur.

When the third approach is used, the application programmer must know (1) the CICS/VS response codes and their meanings, and (2) the symbolic label by which he can refer to the response code. If the Assembler-language or PL/I programmer elects to check for a particular response-code bit pattern, he can access the response code at TCAPCTR. The ANS COBOL programmer who elects to check for a particular response-code bit pattern can access the response code at TCAPCRC. The possible response codes and the conditions to which they correspond are identified in the right-hand columns of Figure 6-7. DFHPC macro instructions for which the conditions are applicable are shown at the left.

Program Services Request by DFHPC Macro Instruction	Condition	Response Code		
		Assembler	ANS COBOL	PL/I
LINK, LOAD SETXIT, CHECK	NORESP (Normal Response)	X'00'	12-0-1-8-9 (PCNORESP)	00000000
LINK, LOAD SETXIT, CHECK	PGMIDER (Program Identification Error)	X'01'	12-1-9 (PCPGMIDER)	00000001
NOTE: The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.				

Figure 6-7. Program Control Response Codes

Note: Because the multipunch codes to be checked in an ANS COBOL program commonly correspond to unprintable characters, an alternative facility is provided in CICS/VS for use by the ANS COBOL programmer. In COBOL the response code can simply be referred to by means of a label, formed as a two-character identification of the CICS/VS management module providing the requested service, followed by the keyword for the condition being checked (for example, PCNORESP). Use

of this approach is illustrated in the examples at the end of this discussion.

If the application programmer wishes to provide for the possibility of failure to find a requested program in the processing program table (PPT) or finding a disabled program in response to DFHPC TYPE=LINK or TYPE=LOAD, the COND operand must be included in the request. This operand causes control to be passed to the user-specified exception-handling routine if the error occurs. If the COND operand is not specified and the error occurs, the requesting program is abnormally terminated with an APCT ABEND code.

The keyword operands that can be used to request tests of the response to a particular request for program services (that is, particular DFHPC macro instruction) are identified in the discussions of the instruction format and operands under "DFHPC Macro Instruction" in Chapter 7. The condition expressed by each keyword is explained in detail and should be referred to by the application programmer when using any of the checking methods described above.

The following examples show how to examine the response code provided by CICS/VS at TCAPCTR (for Assembler language or PL/I) or TCAPCRC (for ANS COBOL) and transfer control tc an appropriate user-written error-handling routine. The alternative approach available to ANS COBOL programmers is also exemplified.

For Assembler language:

```
        DFHPC   TYPE=SETXIT,                                    *
                PROGRAM=MYPROG
        CLI     TCAPCTR,X'00'           NORMAL RESPONSE
        BE      GOOD
        DFHPC   TYPE=ABEND
GOOD    DS      0H
        .
        .
```

For ANS COBOL:

```
        DFHPC   TYPE=SETXIT,                                    *
                PROGRAM=MYPROG
        IF TCAPCRC=' ' THEN GO TO GOOD.  NOTE 12-0-1-8-9 NORESP.
        DFHPC   TYPE=ABEND
GOOD.
        .
        .
```

where the value specified within single quotation marks is an unprintable multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

The alternative approach to response code checking, which is available to ANS COBOL programmers as described earlier, is generally a coding convenience and provides concise code documentation. When this approach is used, the IF statement above is replaced by a statement of the form shown below:

```
        IF PCNORESP THEN GO TC GOOD.
        .
        .
```

For PL/I:

```
       DFHPC   TYPE=SETXIT,                                          *
               PROGRAM=MYPROG
       IF TCAPCTR='00000000'B THEN GO TO GOOD;  /* NORMAL RESPONSE */
GOOD:
       .
       .
       .
```

TIME SERVICES

CICS/VS maintains the current time of day in two formats: binary form at CSACTODB, and packed decimal form at CSATODP. The binary value is updated automatically during task dispatching to reflect the time of day maintained by the operating system. The packed value is updated each time control returns from an operating system WAIT. The accuracy of these values at a given moment depends upon the task mix and the frequency of task switching operations.

Since the time of day maintained by the operating system is changed by the operating system at midnight, CICS/VS must recognize the situation where a "negative" change in the time of day has occurred, and must adjust expiration times maintained by CICS/VS accordingly.

If the optional time adjustment feature of CICS/VS time management is not included in CICS/VS, any change to the operating system time-of-day involving midnight is represented by CICS/VS as a value larger than the previous value (for example, 1:00 a.m. is represented as 2500 hours). If the optional time adjustment feature is included in CICS/VS, any change to the operating system time of day is automatically reflected in the expiration times maintained by CICS/VS.

When the operating system time of day is set to zero at midnight (and the time adjustment feature has been included in CICS/VS), CICS/VS adjusts the expiration times of day it maintains and then resets its time of day to zero. The optional time adjustment feature thus makes it possible for CICS/VS to be operated on a continuous round-the-clock basis.

Time management provides the capability, primarily through interval control and interaction with task control, to control various task functions based on the time of day or on intervals of time. Time services include:

1. Establish the partition/region exit time interval, which is the maximum length of time that CICS/VS voluntarily relinquishes control to the operating system.

2. Provide system stall detection and corrective action based on the expiration of a user-provided time interval, in conjunction with other symptoms of a system stall condition.

3. Provide runaway task detection and corrective action capabilities. A task is considered to be runaway (in an apparent loop) if it executes application program instructions for a user-specified period of time without issuing a request for CICS/VS services that causes transfer of control to the CICS/VS dispatcher, a part of CICS/VS task management.

4. Provide the time of day in binary or packed decimal representation.

5. Provide task synchronization based on time-dependent events.

6. Provide automatic time-ordered task initiation with associated data retention and recovery support.

The services enumerated in items 1 through 3 are CICS/VS system services and require no action on the part of the application programmer. The services enumerated in items 4 through 6 are available to the application programmer through use of the interval control macro instruction (DFHIC).

The application programmer must specify parameter values when using the DFHIC macro instruction. The values can be specified in either of two ways:

- By including the parameters in operands of the DFHIC macro instruction by which time services are requested, or

- By coding instructions that place the parameter values in fields of the TCA prior to issuing the DFHIC macro instruction.

The second of these approaches provides flexibility in that the parameter values of a single DFHIC macro instruction can vary to meet the logic needs of the application program.

The application programmer can check the CICS/VS response to a request for time services as explained under "Test Response to a Request for Time Services." If the programmer does not check for a particular response, and if the condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program. All operands that can be included in the DFHIC macro instruction are discussed in detail under "DFHIC Macro Instruction" in Chapter 7.

TIME-OF-DAY SERVICES (GETIME)

In the course of normal operation, CICS/VS maintains the current time of day in binary form at CSACTODB and in packed decimal form at CSATODP. The binary representation is expressed as a four-byte positive value in hundredths of a second. The packed decimal representation is expressed as a four-byte positive signed value of the form HHMMSSt+ where the seconds are truncated to tenths of a second. The binary value is updated periodically during task dispatching, and the packed decimal value is updated when returning from an operating system WAIT. The accuracy of these values at any given moment depends on the task mix and the frequency of task switching operations.

The application programmer can ensure that one or both of these time-of-day values are updated to a current setting by issuing the

```
DFHIC TYPE=GETIME,
        .
        .
        .
```

macro instruction. This macro instruction causes one or both forms of the time of day to be updated in the CSA and, optionally, places the requested form of the time of day in a four-byte field specified by the application programmer. When the programmer wants the time of day to be returned in a field other than those of the CSA, either the symbolic label of the four-byte field must be specified in the DFHIC TYPE=GETIME macro instruction or the address of the field must be placed in TCAICDA prior to issuing the DFHIC TYPE=GETIME macro instruction.

Note: For performance reasons, it should be recognized that lengthy conversion routines must be executed whenever updating of the packed decimal representation of time of day is requested.

The following example shows how to request that the time of day be placed at the storage locations represented by the symbolic label CLOCK.

```
DFHIC TYPE=GETIME,              REQUEST CURRENT TIME-OF-DAY    *
      FORM=PACKED,              PACKED DECIMAL FORM            *
      TIMADR=CLOCK              SYMBOLIC ADDRESS FOR RESPONSE
```

The following examples show how to request that the time of day be placed in a field selected prior to (and independent of) execution of the DFHIC TYPE=GETIME macro instruction.

For Assembler language:

```
      MVC TCAICDA,=A(CLOCK)     MOVE ADDR FOR RESPONSE TO TCA
      .
      .
      .
      DFHIC TYPE=GETIME,        REQUEST CURRENT TIME-OF-DAY    *
            FORM=PACKED,        PACKED DECIMAL FORM            *
            TIMADR=YES          RESPONSE ADDRESS GIVEN
```

For ANS COBOL:

```
      MOVE CLOCKADR TO TCAICDA.     NOTE MOVE ADDR FOR RESP TO TCA.
      .
      .
      .
      DFHIC TYPE=GETIME,            REQUEST CURRENT TIME-OF-DAY    *
            FORM=PACKED,            PACKED DECIMAL FORM            *
            TIMADR=YES              RESPONSE ADDRESS GIVEN
```

For PL/I:

```
      TCAICDA=ADDR(CLOCK);          /*MOVE ADDR FOR RESP TO TCA*/
      .
      .
      .
      DFHIC TYPE=GETIME,            REQUEST CURRENT TIME-OF-DAY    *
            FORM=PACKED,            PACKED DECIMAL FORM            *
            TIMADR=YES              RESPONSE ADDRESS GIVEN
```

TIME-ORDERED TASK SYNCHRONIZATION (WAIT OR POST)

The task synchronization feature of CICS/VS time management provides the capability to either delay the processing of a requesting task until a specified time occurs or to signal the requesting task when a specified interval of time has elapsed. It also supports the cancellation of a pending time-ordered synchronization event by another task. (See "Time-Ordered Request Cancellation (CANCEL)" later in this section.)

Delay the Processing of a Task (WAIT)

The application programmer can request that the processing of a task be suspended until a specified time of day or for a specified interval of time by issuing the

```
DFHIC TYPE=WAIT,                                                *
      .
      .
      .
```

macro instruction. This macro instruction causes the requesting task to temporarily suspend processing, and to resume control at a specified time of day or after a specified interval of time has elapsed. It supersedes and cancels any previously initiated DFHIC TYPE=POST request for the task.

A numeric value specified in the DFHIC TYPE=WAIT macro instruction (or before this macro instruction is issued) is used by CICS/VS to calculate the time of day the requested time service is to be provided. If the calculated time of day is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have elapsed (occurred) and the requested service is provided immediately. If the calculated time of day is in advance of the current clock time, the requested service is provided when the specified time occurs. If the calculated time of day precedes the current clock time by more than six hours, the requested service is provided the next day at the specified time.

As a means of symbolically identifying the request and any data associated with it, a unique request identification is assigned to each time-ordered request. The application programmer can specify a request identification to be assigned to his DFHIC TYPE=WAIT request. If none is assigned by the programmer, CICS/VS assigns a unique request identification. A request identification should be specified by the application programmer if he wishes to provide another task with the capability of canceling the unexpired WAIT request. (See the discussion of the DFHIC TYPE=CANCEL macro instruction.)

The following example shows how to temporarily suspend the processing of a task for a specified period of time:

```
DFHIC TYPE=WAIT,              DELAY TASK PROCESSING,            *
      INTRVAL=500,            WAIT 5 MINUTES 0 SECONDS          *
      REQID=GXLBZQMR          UNIQUE REQUEST ID
```

The following examples show how to request the suspension of a task until the time of day stored previously in TCAICRT is reached. A request identification previously selected by the user is stored in TCAICQID as a unique identifier for this request for time service.

For Assembler language:

```
      MVC TCAICRT,=PL4'124500'      MOVE 12:45 TO TCA
      MVC TCAICQID,UNIQCODE         UNIQUE REQUEST ID TO TCA
      .
      .
      .
      DFHIC TYPE=WAIT,              DELAY TASK PROCESSING             *
            TIME=YES,               EXPIRATION TIME GIVEN             *
            REQID=YES               UNIQUE ID GIVEN
```

For ANS COBOL:

```
      MOVE 124500 TO TCAICRT.       NOTE MOVE 12:45 TO TCA.
      MOVE UNIQCODE TO TCAICQID.    NOTE UNIQUE REQUEST ID TO TCA.
      .
      .
      .
```

```
      DFHIC TYPE=WAIT,                DELAY TASK PROCESSING          *
            TIME=YES,                 EXPIRATION TIME GIVEN          *
            REQID=YES                 UNIQUE ID GIVEN
```

For PL/I:

```
      TCAICRT=124500;                 /*MOVE 12:45 TO TCA*/
      TCAICQID=UNIQCODE;              /*UNIQUE REQUEST ID TO TCA*/
      .
      .
      DFHIC TYPE=WAIT,                DELAY TASK PROCESSING          *
            TIME=YES,                 EXPIRATION TIME GIVEN          *
            REQID=YES                 UNIQUE ID GIVEN
```

Signal the Expiration of a Specified Time (POST)

The application programmer can request that CICS/VS indicate to the requesting task when a given time has expired by issuing the

```
      DFHIC TYPE=POST,                                               *
            .
            .
```

macro instruction. In response to this macro instruction, CICS/VS makes a timer event control area available to the user for testing. This four-byte storage area is initialized to binary zeros and its address is returned to the requesting task in TCAICTEC.

When CICS/VS determines that the time specified in a DFHIC TYPE=POST macro instruction has expired, byte 0 of the timer event control area is set to a hexadecimal 40 and byte 2 is set to a hexadecimal 80. This form of posting is compatible with the completion code postings performed by the operating systems. The timer event control area can be used as the event contrcl area referred to in a DFHKC TYPE=WAIT macro instruction. (See the discussion of task synchronization under "Task Services.")

The timer event control area provided to the user is not released or altered (except as described above) until one of the following events occurs:

- The task issues a subsequent DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE, or DFHIC TYPE=PUT macro request.
- The task issues a DFHIC TYPE=CANCEL macro request to nullify the DFHIC TYPE=POST request (this releases the storage area occupied by the timer event contrcl area).
- The task terminates, normally or abnormally.

A task can have only one DFHIC TYPE=POST request active at any given time. Any DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE, or DFHIC TYPE=PUT request supersedes and cancels a previously issued DFHIC TYPE=POST request by the task.

Note: The expiration of any CICS/VS time-ordered event is determined by CICS/VS when it is performing its task dispatching function. Therefore, for "posting" to occur, the application programmer must ensure that the task relinquishes control of CICS/VS before each testing of the timer event control area. This can be done directly by issuing the DFHKC TYPE=WAIT or DFHKC TYPE=CHAP macro instruction (see the discussion of task synchronization under "Task Services") or indirectly

by requesting a CICS/VS service which in turn initiates a task service on behalf of the task.

A numeric value specified in or before issuing the DFHIC TYPE=POST macro instruction is used by CICS/VS to calculate the time of day at which the requested time service is to be provided. If the calculated time of day is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have elapsed (occurred) and the requested service is provided immediately. If the calculated time of day is in advance of the current clock time, the requested service is provided when the specified time occurs. If the calculated time of day precedes the current clock time by more than six hours, the requested service is provided the next day at the specified time.

The application programmer can specify a request identification to be assigned to a posting request. If not, CICS/VS assigns a unique request identification, which is returned to the application program in TCAICQID in the form "DFHNNNNN". In either case, the request identification provides a means of symbolically identifying the request.

The following example shows how to request that CICS/VS provide a signal for the task when a specified interval of time has elapsed:

```
DFHIC TYPE=POST,                SIGNAL WHEN INTERVAL PASSES     *
      INTRVAL=30                INTERVAL IS 30 SECONDS
```

The following examples show how to dynamically request that CICS/VS provide a signal for the task when the time of day previously stored in TCAICRT is reached. Since no request identification is specified by the application programmer, CICS/VS automatically assigns one and returns it to the application program at TCAICQID.

For Assembler language:

```
      MVC TCAICRT,PACKTIME      STORE CALCULATED EXPIR TIME
      .
      .
      DFHIC TYPE=POST,          SIGNAL WHEN TIME OCCURS         *
            TIME=YES            EXPIRATION TIME GIVEN
      MVC UNIQCODE,TCAICQID     SAVE CICS/VS UNIQUE REQUEST ID
```

For ANS COBOL:

```
      MOVE PACKTIME TO TCAICRT. NOTE STORE CALC EXPIR TIME.
      .
      .
      DFHIC TYPE=POST,          SIGNAL WHEN TIME OCCURS         *
            TIME=YES            EXPIRATION TIME GIVEN
      MOVE TCAICQID TO UNIQCODE. SAVE CICS/VS UNIQUE REQUEST ID
```

For PL/I:

```
      TCAICRT=PACKTIME;         /*STORE CALCULATED EXPIR TIME*/
      .
      .
      DFHIC TYPE=POST,          SIGNAL WHEN TIME OCCURS         *
            TIME=YES            EXPIRATION TIME GIVEN
      UNIQCODE=TCAICQID;        SAVE CICS/VS UNIQUE REQUEST ID
```

AUTOMATIC TIME-ORDERED TASK INITIATION (INITIATE OR PUT)

This feature of time management allows a task to initiate another task at some future time and, optionally, to pass data to that task.

Task Initiation without Data (INITIATE)

The application programmer can request that another task be initiated at some future time by issuing the

```
DFHIC TYPE=INITIATE,                                          *
      .
      .
      .
```

macro instruction. Through this macro instruction, the application programmer provides the transaction identification of the task to be initiated at some future time and other information pertaining to the task. CICS/VS queues the request until the specified time occurs. Then, as soon as all necessary resources are available (for example, a terminal), the task is initiated. Only one task is initiated if multiple DFHIC TYPE=INITIATE requests for the same transaction and terminal expire at the same time or prior to terminal availability. No data can be passed to the future task by means of the DFHIC TYPE=INITIATE macro instruction. (To do so, see "Task Initiation with Data (PUT)," which follows.) This request supersedes and cancels any previously initiated DFHIC TYPE=POST request by the initiating task.

A numeric value specified in or before issuing the DFHIC TYPE=INITIATE macro instruction is used by CICS/VS to calculate the time of day at which the requested time service is to be provided. If the calculated time of day is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have elapsed (occurred) and the requested service is provided immediately. If the calculated time of day is in advance of the current clock time, the requested service is provided when the specified time occurs. If the calculated time of day precedes the current clock time by more than six hours, the requested service is provided the next day at the specified time.

As stated earlier, a unique request identification is assigned to each time-ordered request as a means of symbolically identifying the request and any data associated with it. The application programmer can specify an identifier for his initiation request, or he can let CICS/VS assign one, in which case it is returned to the application program in TCAICQID in the form "DFHNNNNN".

The application programmer must specify the transaction identification of the future task, either in the DFHIC TYPE=INITIATE macro instruction or by placing it in TCAICTI before issuing the macro instruction. CICS/VS validates the transaction identification by scanning the program control table (PCT). If the specified identifier is not found in the table, CICS/VS does not provide the requested service; a response code is placed at TCAICTR (for Assembler language or PL/I) or at TCAICRC (for ANS COBOL) to indicate that the transaction identification is not valid.

If the future task must communicate with a terminal, the application programmer must also specify a terminal identification, either in the macro instruction or by placing it beforehand in TCAICTID. If it fails to locate the terminal identification in the TCT, CICS/VS validates the terminal identification by scanning the terminal control table (TCT), providing a response code at TCAICTR (for Assembler language or PL/I) or at TCAICRC (for ANS COBOL) without servicing the request.

The following example shows how to request autcmatic initiation of a specified task not associated with a terminal:

```
DFHIC TYPE=INITIATE,          REQUEST TASK INITIATION          *
      INTRVAL=10000,          IN ONE HOUR                      *
      TRANSID=TRNL            TRANSACTION IDENTIFICATION
```

The following examples show how to dynamically request automatic initiation of a task associated with a terminal. The task initiation time, transaction identification, and terminal identification are moved to fields of the TCA before the DFHIC TYPE=INITIATE macro instruction is issued. Since no request identification is specified by the application programmer, CICS/VS automatically assigns one and returns it to the application program at TCAICQID.

For Assembler language:

```
MVC TCAICRT,=PL4'10000',      MOVE ONE HOUR TO TCA
MVC TCAICTI,=CL4'TRN1'        TRANSACTION ID TO TCA
MVC TCAICTID,=CL4'STA5'       TERMINAL ID TO TCA
.
.
.
DFHIC TYPE=INITIATE,          REQUEST TASK INITIATION          *
      INTRVAL=YES,            INTERVAL OF TIME GIVEN           *
      TRMIDNT=YES             TERMINAL ID GIVEN
MVC UNIQCODE,TCAICQID         SAVE CICS/VS UNIQUE REQUEST ID
```

For ANS COBOL:

```
MOVE 10000 TO TCAICRT.        NOTE MOVE ONE HOUR TO TCA.
MOVE 'TRN1' TO TCAICTI.       NOTE TRANSACTION ID TO TCA.
MOVE 'STA5' TO TCAICTID.      NOTE TERMINAL ID TO TCA.
.
.
.
DFHIC TYPE=INITIATE,          REQUEST TASK INITIATION          *
      INTRVAL=YES,            INTERVAL OF TIME GIVEN           *
      TRMIDNT=YES             TERMINAL ID GIVEN
MOVE TCAICQID TO UNIQCODE.    SAVE CICS/VS UNIQUE REQUEST ID
```

For PL/I:

```
TCAICRT=10000;                /*MOVE ONE HOUR TO TCA*/
TCAICTI='TRN1';               /*TRANSACTION ID TO TCA*/
TCAICTID='STA5';              /*TERMINAL ID TO TCA*/
.
.
.
DFHIC TYPE=INITIATE,          REQUEST TASK INITIATION          *
      INTRVAL=YES,            INTERVAL OF TIME GIVEN           *
      TRMIDNT=YES             TERMINAL ID GIVEN
UNIQCODE=TCAICQID;            SAVE CICS/VS UNIQUE REQUEST ID
```

Task Initiation with Data (PUT)

The application programmer can pass data to another task that is to be initiated at some future time by issuing the

```
DFHIC TYPE=PUT,                                                *
          .
          .
          .
```

macro instruction. This macro instruction is used to provide the transaction identification, the location of the data to be stored, and other information applicable to the task to be initiated. CICS/VS stores the data and queues the request until the specified time occurs. As soon as all necessary resources are available (for example, a terminal), the task is initiated. CICS/VS temporary storage management facilities support this facility of time management.

The DFHIC TYPE=PUT macro instruction is used only when data is to be passed to a task to be initiated at some future time. It supersedes and cancels any previously initiated DFHIC TYPE=POST request of the task. If only task initiation at a future time is needed, the DFHIC TYPE=INITIATE macro instruction should be used.

If the task to be initiated is associated with a terminal, the initial DFHIC TYPE=PUT request causes the task to be initiated at the specified time. Subsequent PUTs with the same terminal identification, transaction identification, and expiration time are used to store data for subsequent retrieval by the initiated task. If the task to be initiated is not associated with a terminal, each DFHIC TYPE=PUT request results in a task being initiated at the specified time. That is, only one physical data record can be passed to a task not associated with a terminal. (See "Retrieve Time-Ordered Data.")

Most operands of the DFHIC TYPE=PUT macro instruction are analogous to similar operands of the DFHIC TYPE=INITIATE macro instruction. The discussions of time calculation, request identification, transaction identification, and terminal identification given under "Task Initiation without Data (INITIATE)" apply to DFHIC TYPE=PUT in the same manner as they apply to DFHIC TYPE=INITIATE (see the preceding subsection of this manual). In addition, because the DFHIC TYPE=PUT macro instruction permits data to be passed, the application programmer must specify the symbolic address of the field containing the data. The label may be provided as a parameter of the macro instruction or move the address to TCAICDA prior to issuing the macro instruction.

The data passed to an initiated task must have the standard variable-length format, with the first four bytes containing LL00. LL is a two-byte binary length field (the value of which includes the length of the data plus the first four bytes), and 00 is a two-byte field containing binary zeros.

The following example shows how to request automatic task initiation and/or request that time-ordered data be made available to a task associated with a terminal:

```
DFHIC TYPE=PUT,                 REQUEST TASK INITIATION        *
      TIME=173000,              TIME IS 5:30 PM                *
      TRANSID=TRN2,             TRANSACTION IDENTIFICATION     *
      TRMIDNT=STA3,             TERMINAL IDENTIFICATION        *
      ICDADDR=DATAFLD           DATA ADDRESS
```

The following examples show how to dynamically request automatic task initiation and/or request that time-ordered data be made available to a task associated with a terminal. Values for time, request identification, transaction identification, and terminal identification, as well as the address of data to be passed, are moved to appropriate fields of the TCA before issuing the DFHIC TYPE=PUT macro instruction.

For Assembler language:

```
        MVC TCAICRT,PACKTIME          CALCULATED EXPIR TIME TO TCA
        MVC TCAICQID,UNIQCODE         UNIQUE REQUEST ID TO TCA
        MVC TCAICTI,=CL4'TRN2'        TRANSACTION ID TO TCA
        MVC TCAICTID,=CL4'STA3'       TERMINAL ID TO TCA
        MVC TCAICDA,=A(DATAFLD)       ADDRESS OF DATA TO TCA
        .
        .
        .
        DFHIC TYPE=PUT,               REQUEST TASK INITIATION         *
              TIME=YES,               EXPIRATION TIME GIVEN           *
              TRMIDNT=YES,            TERMINAL ID GIVEN               *
              REQID=YES,              UNIQUE REQUEST ID GIVEN         *
              ICDADDR=YES             DATA ADDRESS GIVEN
```

For ANS COBOL:

```
        MOVE PACKTIME TO TCAICRT.     NOTE CALC EXPIR TIME TO TCA
        MOVE UNIQCODE TO TCAICQID.    NOTE UNIQUE REQUEST ID TO TCA.
        MOVE 'TRN2' TO TCAICTI.       NOTE TRANSACTION ID TO TCA.
        MOVE 'STA3' TO TCAICTID.      NOTE TERMINAL ID TO TCA.
        MOVE DATADDR TO TCAICDA.      NOTE ADDRESS OF DATA TO TCA.
        .
        .
        .
        DFHIC TYPE=PUT,               REQUEST TASK INITIATION         *
              TIME=YES,               EXPIRATION TIME GIVEN           *
              TRMIDNT=YES,            TERMINAL ID GIVEN               *
              REQID=YES,              UNIQUE REQUEST ID GIVEN         *
              ICDADDR=YES             DATA ADDRESS GIVEN
```

For PL/I:

```
        TCAICRT=PACKTIME;             /*CALC EXPIR TIME TO TCA*/
        TCAICQID=UNIQCODE;            /*UNIQUE REQUEST ID TO TCA*/
        TCAICTI='TRN2';               /*TRANSACTION ID TO TCA*/
        TCAICTID='STA3';              /*TERMINAL ID TO TCA*/
        TCAICDA=ADDR(DATAFLD);        /*ADDRESS OF DATA TO TCA*/
        .
        .
        .
        DFHIC TYPE=PUT,               REQUEST TASK INITIATION         *
              TIME=YES,               EXPIRATION TIME GIVEN           *
              TRMIDNT=YES,            TERMINAL ID GIVEN               *
              REQID=YES,              UNIQUE REQUEST ID GIVEN         *
              ICDADDR=YES             DATA ADDRESS GIVEN
```

RETRIEVE TIME-ORDERED DATA (GET)

Tasks can retrieve expired time-ordered data by issuing the

```
        DFHIC TYPE=GET,                                               *
              .
              .
              .
```

macro instruction. Only data from an expired DFHIC TYPE=PUT request can be accessed using the DFHIC TYPE=GET macro instruction. To retrieve data stored by use of a DFHIC TYPE=PUT request, the DFHIC TYPE=GET macro instruction must be used.

When time-ordered data is to be retrieved by means of a DFHIC TYPE=GET macro instruction, the application programmer may specify the address of a storage area into which the data is to be placed. The address is specified either by including the address in the macro instruction or by storing it in TCAICDA prior to issuing the macro instruction. In either case, the storage area must be large enough to contain the four-byte length field (LLØØ) at the beginning of the data record as well as the data portion of the record. If the application programmer does not select a storage area, CICS/VS automatically acquires an area of sufficient size and returns the address of that area in TCAICDA.

Each originating DFHIC TYPE=PUT request provides the transaction identification of the task to receive the data, and if applicable, symbolically identifies the terminal associated with the task's operation. When CICS/VS services a DFHIC TYPE=PUT request, it does so in two steps; it first queues the request for automatic task initiation at a specified time and then stores the data. When the specified time occurs, the task is ready to be initiated, and the stored data is then available for retrieval.

A task not associated with a terminal that is initiated as a result of an expired DFHIC TYPE=PUT request can access only the single physical data record associated with the original request. It does this by issuing one DFHIC TYPE=GET macro instruction. The storage occupied by the data associated with the task is released upon execution of the DFHIC TYPE=GET request, or upon termination of the task (normally or abnormally) if no DFHIC TYPE=GET macro instruction is executed prior to termination.

A task associated with a terminal that is initiated as the result of an expired DFHIC TYPE=PUT request, or that is active at the time of expiration of a DFHIC TYPE=PUT request, can access all data records associated with expired DFHIC TYPE=PUT macro requests having the same transaction identification and terminal identification. Therefore, a task associated with a terminal can retrieve all data made available to the terminal and the task up to the current time by issuing consecutive DFHIC TYPE=GET requests. Expired data records are presented to the task upon request in expiration time sequence. The storage occupied by the single data record associated with a DFHIC TYPE=PUT request is released after the data has been retrieved by a DFHIC TYPE=GET request or upon termination of the program if the task terminates (normally or abnormally) without retrieving the data. Data passed in subsequent expired DFHIC TYPE=PUT requests specifying the same terminal identification and transaction identification can be retrieved in response to DFHIC TYPE=GET requests by the same task if that task is still active at their expiration times. Otherwise, such a DFHIC TYPE=PUT request causes a new task to be initiated.

When all passed data for which specified times have expired has been retrieved, CICS/VS provides an end-of-data response at TCAICTR (for Assembler language or PL/I) or TCAICRC (for ANS COBOL) in response to a DFHIC TYPE=GET macro instruction.

The following example shows how to request retrieval of a time-ordered data record into a data area specified in the request:

```
DFHIC TYPE=GET,                  RETRIEVE TIME-ORDERED DATA      *
      ICDADDR=DATAFLD            USER-PROVIDED DATA AREA
```

The following examples show how to dynamically request retrieval of a time-ordered data record. The address of the storage area reserved for the data record is placed in TCAICDA prior to the issuance of the DFHIC TYPE=GET macro instruction.

For Assembler language:

```
        MVC TCAICDA,=A(DATAFLD)          DATA FIELD ADDR TO TCA
        .
        .
        .
        DFHIC TYPE=GET,                  RETRIEVE TIME-ORDERED DATA      *
              ICDADDR=YES                DATA FIELD ADDRESS GIVEN
```

For ANS COBOL:

```
        MOVE DATADDR TO TCAICDA.         NOTE DATA FIELD ADDR TO TCA.
        .
        .
        .
        DFHIC TYPE=GET,                  RETRIEVE TIME-ORDERED DATA      *
              ICDADDR=YES
```

For PL/I:

```
        TCAICDA=ADDR(DATAFLD) ;          /*DATA FIELD ADDR TO TCA*/
        .
        .
        .
        DFHIC TYPE=GET,                  RETRIEVE TIME-ORDERED DATA      *
              ICDADDR=YES
```

TIME-ORDERED REQUEST CANCELLATION (CANCEL)

The application programmer can request that a previously issued time-ordered service request (DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE, or DFHIC TYPE=PUT) be canceled by issuing the

```
        DFHIC TYPE=CANCEL,                                               *
              .
              .
              .
```

macro instruction. The effect of the cancellation is dependent on whether a request identification is specified for the DFHIC TYPE=CANCEL request and on the type of service request being canceled.

Cancel an Interval Control POST Request

A DFHIC TYPE=POST request can be canceled by the originating task or by another task through use of the DFHIC TYPE=CANCEL macro instruction.

When the originating task cancels a DFHIC TYPE=POST request, no request identification should be specified for the cancellation request. This cancellation request can be made either before or after expiration of the original request. In either case, the storage reserved for the timer event control area is released, and all references to the original request are removed from the system.

When a task other than the originating task cancels a DFHIC TYPE=POST request, the request identification of that request must be specified. The effect of the cancellation is the same as an early expiration of the original DFHIC TYPE=POST request. That is, the timer event control area for the originating task is posted as though the original expiration time had been reached.

Cancel an Interval Control WAIT Request

A DFHIC TYPE=WAIT request can only be canceled prior to its expiration, and only by a task other than the task that issued the DFHIC TYPE=WAIT (the originating task is suspended for the duration of the request). The request identification of the suspended task must be specified. The effect of the cancellation is the same as an early expiration of the original DFHIC TYPE=WAIT or DFHKC TYPE=CHAP request. That is, the originating task resumes control (based on its normal dispatching priority) as though the original expiration time had been reached.

Cancel an Interval Control INITIATE or PUT Request

A request identification must be specified when the DFHIC TYPE=CANCEL macro instruction is used to cancel a DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request. The effect of the cancellation is to remove the original request from the system, treating the original request as though it had never been made. The cancellation request is effective only prior to expiration of the original request.

INPUT/OUTPUT ERROR RETRY CAPABILITY (RETRY)

When the response to a DFHIC TYPE=GET macro instruction indicates an I/O error, the application programmer can issue the

```
DFHIC TYPE=RETRY,                                              *
      .
      .
      .
```

macro instruction, requesting that CICS/VS retry the retrieval operation. CICS/VS attempts to retrieve the data record whose symbolic eight-character identification is specified at TCAICQID, and place it into the data area specified at TCAICDA. These fields are preset by CICS/VS at the time the I/O error response was returned to the application program.

TEST RESPONSE TO A REQUEST FOR TIME SERVICES

After issuing a request for time services, the application programmer can check the CICS/VS response to determine subsequent processing that should be carried out. One step in doing so is to specify the symbolic addresses of user-written exception-handling routines, any of which may be executed as a result of the check. The addresses can be specified in any of three ways:

- Include the symbolic addresses in operands of the DFHIC macro instruction by which time service is requested.
- Include the symbolic addresses in operands of a

  ```
  DFHIC TYPE=CHECK
        .
        .
  ```

  macro instruction immediately following the DFHIC macro instruction by which the service is requested.
- Include instructions immediately following the DFHIC macro instruction that examine the response code set automatically by

CICS/VS and transfer control to an exception-handling routine accordingly.

The general discussion under "Test Response to a Request for File Services" applies to time services as well. The Assembler-language or PL/I programmer can access time response codes at TCAICTR; the American National Standard (ANS) COBOL programmer can access time response codes at TCAICRC. The possible response codes and the conditions to which they correspond are identified in the right-hand columns of Figure 6-8. In addition, the ANS COBOL programmer can refer to the response codes by means of symbolic labels (ICNORESP, ICEXPIRD, and so on) to cause specific response code patterns to be checked without specifying those patterns in his program. (See the examples at the end of this discussion.) DFHIC macro instructions for which the conditions are applicable are shown at the left.

Time Services Request by DFHIC Macro Instruction	Condition	Response Code		
		Assembler	ANS COBOL	PL/I
ALL	NORESP (Normal Response)	X'00'	12-0-1-8-9 (ICNORESP)	00000000
GET, CHECK	ENDDATA (End of Data Condition)	X'01'	12-1-9 (ICENDDATA)	00000001
PUT, GET, RETRY, CHECK	IOERROR (Input/Output Error)	X'04'	12-4-9 (ICIOERROR)	00000100
INITIATE, PUT, CHECK	TRNIDER (Transaction Identification Error)	X'11'	11-1-9 (ICTRNIDER)	00010001
INITIATE, PUT, CHECK	TRMIDER (Terminal Identification Error)	X'12'	11-2-9 (ICTRMIDER)	00010010
GET, CHECK	TSINVLD (No Temporary Storage Support)	X'14'	11-4-9 (ICTSINVLD)	00010100
WAIT, POST, CHECK	EXPIRD (Expired)	X'20'	11-0-1-8-9 (ICEXPIRD)	00100000
GET, CANCEL, RETRY, CHECK	NOTFND (Not Found)	X'81'	12-0-1 (ICNOTFND)	10000001
ALL	INVREQ (Invalid Request)	X'FF'	12-11-0-7-8-9 (ICINVREQ)	11111111
ALL	ERROR (Any Response Other Than NORESP)	See Note 2	See Note 2	See Note 2

NOTES:
1. The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.
2. The test for the ERROR response is satisfied by a not equal condition; that is, not X'00', not 12-0-1-8-9, or not 00000000 for Assembler, ANS COBOL, and PL/I, respectively.

Figure 6-8. Interval Control Response Codes

If the application programmer does not check for a particular response to his service request, and if the exception condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program.

The keyword operands that can be used to request tests of the response to a particular request for time services (that is, a particular DFHIC macro instruction) are identified in the discussions of the instruction format and operands under "DFHIC Macro Instruction." The condition expressed by each keyword is explained in detail and

should be referred to by the application programmer when using any of the checking methods described above.

The following examples show how to examine the response code provided by CICS/VS at TCAICTR (for Assembler language or PL/I) or TCAICRC (for ANS COBOL) and transfer control to the appropriate user-written exception-handling routine. The alternative approach available to ANS COBOL programmers is also exemplified.

For Assembler language:

```
        DFHIC   TYPE=GET,
                ICDADDR=DATAFLD
        CLI     TCAICTR,X'00'           NORMAL RESPONSE
        BE      GOOD
        DFHPC   TYPE=ABEND
GOOD    DS      0H
        .
        .
        .
```

For ANS COBOL

```
        DFHIC   TYPE=GET,
                ICDADDR=DATAFLD
        IF TCAICRC=' ' THEN GO TO GOOD.  NOTE 12-0-1-8-9 NORESP.
        DFHPC   TYPE=ABEND
GOOD
        .
        .
        .
```

where the value specified within single quotation marks is an unprintable multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

The alternative approach to response code checking available to ANS COBOL programmers is generally a coding convenience and provides concise code documentation. When this approach is used, the IF statement above is replaced by a statement of the form shown below:

```
        IF ICNORESP THEN GO TO GOOD.
        .
        .
        .
```

For PL/I:

```
        DFHIC   TYPE=GET,
                ICDADDR=DATAFLD
        IF TCAICTR='00000000'B THEN GO TO GOOD;/*NORMAL RESPONSE*/
        DFHPC   TYPE=ABEND
GOOD:
        .
        .
        .
```

TASK SERVICES

Task management provides the capability to process transactions (tasks) concurrently. Transactions are scheduled, through task control, and processed according to priorities assigned by the user. Control

of the central processing unit (CPU) is given to the highest priority task that is ready to be processed. Control of the CPU is returned to the operating system when no further work can be done by CICS/VS or by user-written application programs.

When a transaction is initiated in CICS/VS, task control dynamically allocates storage for the task ccntrol area (TCA), places the task on the dispatching priority queue, obtains the program identification of the program initially required to process the task from the program control table (PCT), and transfers control to program control.

The Task Management macro instruction (DFHKC) is used to request any of the following services:

- Initiate a task
- Change the priority of a task
- Synchronize a task
- Synchronize the use of a resource by a task
- Purge a task on system overload

The application programmer must specify parameter values when using the DFHKC macro instruction. The values can be specified in either of two ways:

- By including the parameters in operands of the DFHKC macro instruction by which task control services are requested, or
- By coding instructions that place the parameter values in fields of the TCA prior to issuing the DFHKC macro instruction

The second method adds flexibility by letting the programmer vary the parameter values of a single DFHKC macro instruction to meet the needs of a given program.

INITIATE A TASK (ATTACH)

Task initiation within CICS/VS is invoked by issuing the

```
DFHKC TYPE=ATTACH,                                              *
      .
      .
```

macro instruction. This macro instruction causes task control to obtain the task control area (TCA) for a task and insert the task in the dispatching priority queue according to the overall transaction processing priority of the task. This macro instruction is intended to be used by other CICS/VS control modules, but it is also available for use by the application programmer to initiate additional tasks. Any additional tasks initiated by the application programmer must terminate themselves through use of the program control DFHPC TYPE=RETURN macro instruction.

Note: Here and elsewhere in this chapter, the general format of the macro instruction is not printed in full. The general format and all operands are explained in detail in Chapter 7.

Most tasks running under CICS/VS are initiated (attached) at a terminal and are thus associated with a terminal. Tasks initiated by CICS/VS management programs (for example, automatic task initiation by transient data control) may or may not be associated with a terminal.

The contents of TCAFCAAA varies depending upon whether the attached task is associated with a terminal, as discussed in "task control area (TCA)" of Chapter 2.

The number of tasks that can be active within the system at a given time is limited by the availability of main storage and/or by the "maximum number of tasks" control established by the system programmer at system generation or initialization. A new task is not initiated by CICS/VS unless sufficient main storage is available to process it. Instead, the request to initiate a task is queued (stored) until sufficient main storage becomes available. Tasks initiated by CICS/VS management modules (for example, terminal control) are subject to the maximum number of tasks limitation. Application program requests for attachment of tasks are not subject to this limitation and therefore are allowed to exceed the maximum.

If the DFHKC TYPE=ATTACH macro instruction is used by the application programmer, he must provide the facility control area address and transaction identification required by CICS/VS to initiate a new task. The address and identification can be specified in two ways.

1. By coding two instructions that assign a facility control area address to TCAKCFA and a transaction identification to TCAKCTI prior to issuing the DFHKC TYPE=ATTACH macro instruction, or

2. By including the FCADDR=symbolic address operand and TRANSID=symbolic name operand in the DFHKC TYPE=ATTACH macro instruction, which then stores the assigned values in TCAKCFA and TCAKCTI, respectively.

For all transactions associated with a terminal, the facility control area address in TCAKCFA is the address of the TCTTE for the terminal. This address provides access to control information necessary for communication between the program and the terminal. The first byte is an X'01'. If the attaching task owns a terminal, ownership of that terminal (but of no other terminal) may be passed in TCAKFCA.

If a task is not associated with a terminal, the facility control area address can serve as a pointer to additional facility control information required for execution of the task. For example, it can be the address of an entry in the destination control table (DCT) that is associated with a hardware resource (for example, a data set).

The transaction identification is used only for the current ATTACH; it is not carried in the TCA for the duration of the task.

The specified task is not attached if the transaction identification is not in the PCT or the program name is not in the Processing Program Table (PPT). If this situation exists or the attached task ABENDs, a message is sent to the terminal operator, but the attaching task is not notified of the condition. Therefore, the DFHKC TYPE=ATTACH macro instruction must be used with extreme caution by the application programmer.

Although the application programmer has the capability of attaching a task directly to a terminal by means of the DFHKC TYPE=ATTACH macro, this procedure is not recommended. Rather, any of the following approaches should be used:

- Automatic task initiation through transient data management
- Automatic task initiation through time management (interval control program)

- Identification of the transaction identification to be used with the next input message from the terminal by means of a DFHPC TYPE=RETURN macro instruction.

The flowchart in Figure 6-9 shows Task A attaching Task B and synchronizing the processing steps of both tasks through use of the facility control address passed to the newly created task at attach time. Since Task B is a nonterminal-oriented task, it is unable to use terminal control macro instructions. FCADDR specifies the address of Task A's TCA; ECB1 and ECB2 are fields in the TWA for Task A.

Figure 6-9 includes steps labeled "POST ECB". Posting an ECB entails setting on the appropriate bit in the ECB, which is a 4-byte field. In OS/VS, the bit to be set on (that is, set to '1') is bit 1 of byte 0; in DOS/VS, it is bit 0 of byte 2. The following examples show how to set bits on for each programming language. These examples set on both the bit required for OS/VS and that required for DOS/VS, so they may be used for OS/VS and DOS/VS systems.

For Assembler language:

```
ECB1  DC   F'0'
      MVC  ECB1(3),=X'400080'
```

For ANS COBOL:

```
77 ECB1 USAGE COMPUTATIONAL PICTURE S9(8) VALUE IS ZERO.
    .
    .
    .
PROCEDURE DIVISION.
   COMPUTE ECB1 = 2 ** 15 + 2 ** 30.
```

For PL/I:

```
DCL ECB1 BIT(32) ALIGNED INIT('0'B);
ECB1 = '01000000000000001'B;
```

The DFHPC TYPE=RETURN macro instruction can be used to terminate any tasks initiated by the application programmer through use of the task control DFHKC TYPE=ATTACH macro instruction.

The following example illustrates the coding required to statically provide a facility control area address and transaction identification:

```
DFHKC TYPE=ATTACH,          INITIATE NEW TASK                    *
      FCADDR=FACCTL,        USER'S FCA ADDRESS                   *
      TRANSID=TRN1          TRANSACTION IDENTIFICATION
```

The following examples illustrate the coding required to dynamically provide a facility control area address and transaction identification.

For Assembler language:

```
MVC TCAKCTI,=CL4'TRN1'      TRANSACTION IDENTIFICATION
MVC TCAKCFA,=A(FACCTL)      USER'S FCA ADDRESS
```

```
          .
          .
          .
      DFHKC TYPE=ATTACH           INITIATE NEW TASK
```

For ANS COBOL:

```
      MOVE 'TRN1' TO TCAKCTI.     NOTE TRANSACTION IDENTIFICATION.
      MOVE FACADR TO TCAKCFA.     NOTE USER'S FCA ADDRESS.
           .
           .
           .
      DFHKC TYPE=ATTACH           INITIATE NEW TASK
```

For PL/I:

```
      TCAKCTI='TRN1';             /*TRANSACTION IDENTIFICATION*/
      TCAKCFA=FACADR;             /*USER'S FCA ADDRESS*/
           .                      /*FACADR IS A POINTER VARIABLE*/
           .
           .
      DFHKC TYPE=ATTACH           INITIATE NEW TASK
```

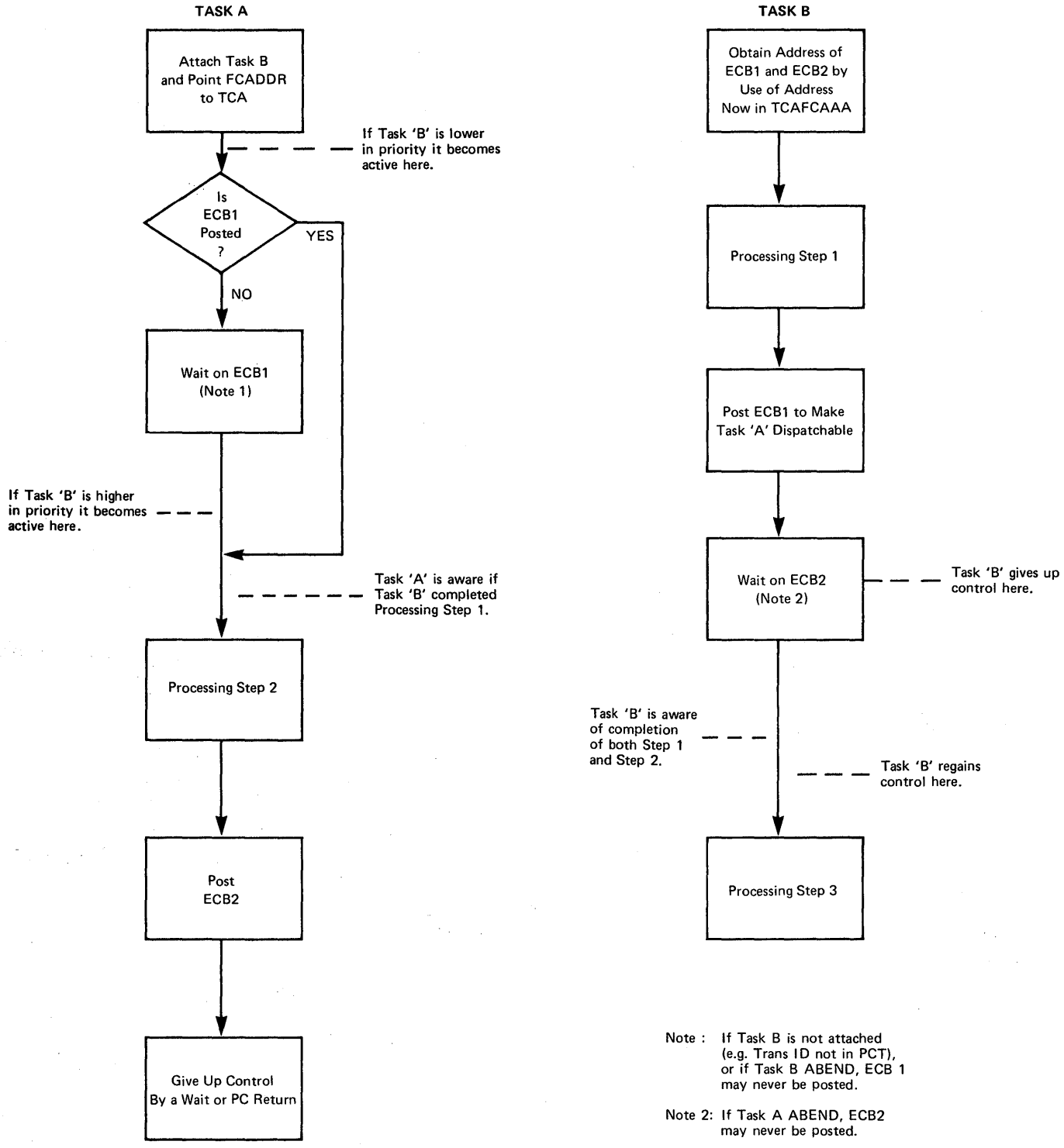


Figure 6-9. Task Synchronization under CICS/VS

RESCHEDULE A 3650 TASK (SCHEDULE)

During operation with a 3650 system, it may be necessary to reschedule an automatic- or time-initiated task rejected because of

insufficient resources in the 3651. Such a task can be rescheduled for operation by issuing the

```
DFHKC TYPE=SCHEDULE
```

macro instruction. This macro instruction causes the task to be rescheduled and another attempt is made to initiate execution.

CHANGE PRIORITY OF A TASK (CHAP)

The overall transaction processing priority of a task is the sum of related transaction, terminal, and operator priorities as specified or established by default at system generation. This priority determines the position of the task in the dispatching priority queue and, therefore, its scheduling under CICS/VS. The priority of an existing task can be changed by issuing the

```
DFHKC TYPE=CHAP,                                                *
      .
      .
```

macro instruction. The specified priority value must be in the range from 0 through 255, where 255 represents the highest priority. This task is placed below all other tasks of equal or higher priority in the dispatching priority queue.

The application programmer can include the PRTY=priority value operand in the DFHKC TYPE=CHAP macro instruction to assign a new dispatching priority to a task. Alternatively, the programmer can assign a priority value to the dispatching priority field (TCATCDP) prior to issuing the DFHKC TYPE=CHAP macro instruction.

A task can voluntarily relinquish control to all tasks of equal or higher priority by issuing a

```
DFHKC TYPE=CHAP
```

macro instruction. No priority value is specified, and the current priority value of the task as stored in TCATCDP is not changed. However, the fact that the macro instruction is issued permits control to be transferred from the task issuing the instruction to an equal or higher priority task within CICS/VS. This capability is designed particularly for compute-bound tasks which, by continually demanding inordinate amounts of CPU time, can significantly affect overall system performance.

The following example shows how to statically assign a new task dispatching priority value:

```
DFHKC TYPE=CHAP,        CHANGE PRIORITY OF THIS TASK            *
      PRTY=255          NEW PRIORITY VALUE
```

The following examples illustrate the coding required to assign a dynamically selected priority value. This value can be specified as a binary, decimal, or hexadecimal number, depending on the programming language used.

For Assembler language:

```
      MVI TCATCDP,X'FF'       ASSIGN NEW PRIORITY VALUE
          .
          .
          .
```

```
      DFHKC TYPE=CHAP          CHANGE PRIORITY OF THIS TASK
```

For ANS COBOL:

```
      MOVE '*' TO TCATCDP.     NOTE ASSIGN NEW PRIORITY VALUE.
                               '*' IS A MULTIPUNCH  12-11-0-7-8-9.
           .
           .
           .
      DFHKC TYPE=CHAP          CHANGE PRIORITY OF THIS TASK
```

For PL/I:

```
      TCATCDP=255;             /*ASSIGN NEW PRIORITY VALUE*/
           .
           .
      DFHKC TYPE=CHAP          CHANGE PRIORITY OF THIS TASK
```

SYNCHRONIZE A TASK (WAIT)

The application programmer can synchronize a task with the completion of an event or one of a list of events initiated by the same task or by another task, or voluntarily relinquish control to a task of higher dispatching priority, by issuing the

```
      DFHKC TYPE=WAIT,                                              *
            .
            .
```

macro instruction. In the first case, this macro instruction provides a method of directly relinquishing control to some other task until the event being waited on is completed. In the latter case, the task remains dispatchable. That is, execution of the task is resumed if no task of higher priority is ready to be processed.

The application programmer must specify the circumstances under which synchronization of a task is to occur by including the DCI=keyword operand (dispatch control indicator) in the DFHKC TYPE=WAIT macro instruction.

If the task is to be synchronized with the completion of a single event or an event of a list of events, the application programmer must specify the symbolic address of either the single event control area or the list of event control areas. The address can be specified by including the ECADDR=symbolic address operand in the DFHKC TYPE=WAIT macro instruction, or by coding a single instruction that places the event control address in TCATCEA prior to issuing the DFHKC TYPE=WAIT macro instruction. In either case, the referenced event control area(s) must conform to the format and standard posting conventions associated with the operating system (for example, ECBs in VS1 or VS2, CCBs in DOS/VS). An event control area can also be the timer event control area referred to in a DFHIC TYPE=POST macro instruction. (See the discussion of task synchronization under "Time Services.")

Synchronize a Task with a Single Event

The DFHKC TYPE=WAIT,DCI=SINGLE macro instruction is used by the application programmer to synchronize a task with the completion of a single event initiated by the same task or by another task.

The following example shows how to synchronize a task with a single event, statically providing the symbolic address of the appropriate event control area:

```
DFHKC TYPE=WAIT,          RELINQUISH CONTROL OF CICS/VS      *
      DCI=SINGLE,         WAIT ON SINGLE EVENT               *
      ECADDR=EVENTCTL     ADDRESS OF EVENT CONTROL AREA
```

The following examples show how to synchronize a task with a single event, dynamically providing the symbolic address of the appropriate event control area.

For Assembler language:

```
       ST    SINGADDR,TCATCEA      PLACE SYMBOLIC ADDRESS IN TCA
        .
        .
        .
       DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS/VS        *
             DCI=SINGLE            WAIT ON SINGLE EVENT
```

For ANS COBOL:

```
       MOVE SINGADDR TO TCATCEA.   NOTE PLACE SYMBOLIC ADDR IN TCA.
        .
        .
        .
       DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS/VS        *
             DCI=SINGLE            WAIT ON SINGLE EVENT
```

For PL/I:

```
       TCATCEA=SINGADDR;           /*PLACE SYMBOLIC ADDRESS IN TCA*/
        .                          /*SINGADDR IS A POINTER VARIABLE*/
        .
        .
       DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS/VS        *
             DCI=SINGLE            WAIT ON SINGLE EVENT
```

Synchronize a Task with One of a List of Events

The DFHKC TYPE=WAIT,DCI=LIST macro instruction is used by the application programmer to synchronize a task with the completion of one event of a list of events. This list consists of a series of contiguous four-byte fields, each field containing the symbolic address of a single event control area. The last four-byte field of the list contains binary ones, hexadecimal Fs, or the card code (multipunch) 12-11-0-7-8-9.

The following example shows how to synchronize a task with one of a list of events, statically providing the symbolic address of the appropriate list of events:

```
       DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS/VS        *
             DCI=LIST,             WAIT ON A LIST OF EVENTS             *
             ECADDR=TOPOLIST       ADDRESS OF LIST OF EVENTS
```

The following examples show how to synchronize a task with one of a list of events, dynamically providing the symbolic address of the appropriate list of events.

For Assembler language:

```
       ST   LISTADDR,TCATCEA       PLACE SYMBOLIC ADDRESS IN TCA
        .
        .
        .
       DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS/VS        *
             DCI=LIST              WAIT ON A LIST OF EVENTS
```

For ANS COBOL:

```
       MOVE LISTADDR TO TCATCEA.  NOTE PLACE SYMBOLIC ADDR IN TCA.
         .
         .
         .
       DFHKC TYPE=WAIT,           RELINQUISH CONTROL OF CICS/VS        *
             DCI=LIST             WAIT ON A LIST OF EVENTS
```

For PL/I:

```
       TCATCEA=LISTADDR;          /*PLACE SYMBOLIC ADDRESS IN TCA*/
         .                        /*LISTADDR IS A POINTER VARIABLE*/
         .
       DFHKC TYPE=WAIT,           RELINQUISH CONTROL OF CICS/VS        *
             DCI=LIST             WAIT ON A LIST OF EVENTS
```

Relinquish Control to a Task of Higher Priority

The DFHKC TYPE=WAIT,DCI=DISP macro instruction is used by the application programmer to voluntarily relinquish control to a task of higher dispatching priority. Control is returned to the task issuing the macro instruction if no other task of a higher priority is ready to be processed.

When binary synchronous communication lines are part of the user's configuration, these lines may time out if excessive CPU time is required by an application program. One way to avoid this condition is to include one or more DFHKC TYPE=WAIT,DCI=DISP macro instructions in the application program to voluntarily relinquish control before the line time-out can occur.

The following example shows how to voluntarily relinquish control to a task of higher dispatching priority:

```
       DFHKC TYPE=WAIT,           RELINQUISH CONTROL OF CICS/VS        *
             DCI=DISP             AND REMAIN DISPATCHABLE
```

Note: The DFHKC TYPE=WAIT macro instruction differs from a TYPE=CHAP macro instruction that does not indicate a priority in that the former relinquishes control to only a task of higher priority, while the latter may relinquish control to a task of either equal or higher priority.

SINGLE-SERVER RESOURCE SYNCHRONIZATION (ENQ/DEQ)

In the CICS/VS environment, where tasks are processed concurrently, it is sometimes desirable to protect a given resource from concurrent use by multiple tasks. In effect, the resource can be treated as serially reusable. To provide this resource protection, an installation convention must be established for all application programmers to follow. The convention is based on use of the

```
       DFHKC TYPE=ENQ,                                               *
               .
               .
```

macro instruction, identifying the resource by a symbolic address or a character-string argument. When executed, this macro instruction causes further execution of the task issuing the instruction to be synchronized with the availability of the specified resource; control is returned to the task when the resource is available. When all

programs accessing a resource adhere to the convention of enqueuing upon the resource, the resource is afforded "single-server" protection.

When a single-server resource is being used by a task and other tasks concurrently enqueue upon the same resource, the first task to issue the DFHKC TYPE=ENQ macro instruction receives the resource when it becomes available. The other tasks obtain the resource, in turn, in the order in which they enqueue upon it.

The application programmer can release a resource currently enqueued upon by the task from single-server protection request for a resource by issuing the

```
DFHKC TYPE=DEQ,                                                    *
        .
        .
```

macro instruction. If a task enqueues upon a resource but does not dequeue it, task control automatically dequeues the single-server protection request upon termination of the task.

When issuing the DFHKC TYPE=ENQ macro instruction, the application programmer must identify the single-server resource he is enqueuing upon by one of the following methods:

- Specify a symbolic main storage address that represents the single-server resource. The application programmer must provide the symbolic main storage address in the DFHKC TYPE=ENQ macro instruction or by coding instructions (prior to issuing the DFHKC TYPE=ENQ macro instruction) that place the address in the low-order three bytes of TCATCQA, a four-byte field. He must place binary zeros in the high-order byte.

- Specify a symbolic main storage address that contains a unique character-string argument (for example, an employee name) that represents the single-server resource. The unique argument may be up to 255 bytes in length, beginning at the location pointed to by the contents of the specified address. The application programmer must provide the symbolic main storage address and the length in the DFHKC TYPE=ENQ macro instruction or by coding instructions (prior to issuing the DFHKC TYPE=ENQ macro instruction) that place the symbolic address in the low-order three bytes of TCATCQA, a four-byte field, and the length (in bytes) in the high-order byte. CICS/VS task control makes a copy of this pointer in its storage for use in controlling the resource.

When issuing the DFHKC TYPE=DEQ macro instruction, the application programmer must identify the resource he is dequeuing by the method that was used in enqueuing. The ANS COBOL programmer may find it convenient to use the program control DFHPC TYPE=COBADDR macro instruction (see below) if preloading of the address is desired.

The following examples show how to enqueue upon a single-server resource using method 1, above.

For Assembler language:

```
         COPY DFHCSADS
CSAWABA  DS   F
       .
       .
       .
         DFHKC TYPE=ENQ,                ENQ ON SINGLE-SERVER RESOURCE   *
               QARGADR=CSAWABA          SPECIFY SYMBOLIC ADDRESS
```

```
                OR

      LA  WORKREG,CSAWABA
      ST  WORKREG,TCATCQA
      .
      .
      DFHKC TYPE=ENQ
```

For ANS COBOL:

```
      01  DFHCSADS COPY DFHCSADS.
          02  CSAWABA PICTURE X(50).
      .
      .
      .
      MOVE ZEROS TO TCATCQA
      DFHKC TYPE=ENQ,                  ENQ ON SINGLE-SERVER RESOURCE   *
            QARGADR=CSAWABA            SPECIFY SYMBOLIC ADDRESS

                  OR

      DFHPC TYPE=COBADDR,                                              *
            LABEL=CSAWABA
      MOVE TCAPCLA TO TCATCQA.
      .
      .
      .
      DFHKC TYPE=ENQ
```

For PL/I:

```
      %INCLUDE DFHCSADS;
      DECLARE 1 DFHEXCSA BASED (CSACBAR),
          2 FILLER CHAR (512),
          2 CSAWABA CHAR (50);
      .
      .
      .
      DFHKC TYPE=ENQ,                  ENQ ON SINGLE-SERVER RESOURCE   *
            QARGADR=CSAWABA            SPECIFY SYMBOLIC ADDRESS

                  OR

      TCATCQA=ADDR(CSAWABA);
      .
      .
      .
      DFHKC TYPE=ENQ
```

The following examples show how to enqueue upon a single-server resource using method 2. The resource to be enqueued upon is identified by the nine-character social security number in a field labeled SOCSECNO. Task control makes a copy of this field for its use in controlling the resource.

For Assembler language:

```
      DFHKC TYPE=ENQ,                                                  *
            QARGADR=SOCSECNO,                                          *
            QARGLNG=9
            OR
```

```
       LA  WORKREG,SOCSECNO
       ST  WORKREG,TCATCQA
       MVI TCATCQA,X'09'
       .
       .
       .
       DFHKC TYPE=ENQ
```

For ANS COBOL:

```
       DFHKC TYPE=ENQ,                                          *
             QARGADR=SOCSECNO,                                  *
             QARGLNG=9
```

For PL/I:

```
       DFHKC TYPE=ENQ,                                          *
             QARGADR=SOCSECNO,                                  *
             QARGLNG=9

                   OR

       %INCLUDE DFHTCADS;
       DECLARE 1 DFHEXTCA BASED (TCACBAR),
           2 FILLER CHAR (20),
           2 TCATCQAL BIT(8);
       .
       .
       TCATCQA=ADDR(SOCSECNO);
       TCATCQAL='00001001'B;
       .
       .
       DFHKC TYPE=ENQ
```

Substituting "DEQ" for "ENQ" in these examples illustrates the ways in which the application programmer can release single-server protection from a resource pricr to termination of the associated task.

DECLARE THE PURGEABILITY OF A TASK ON SYSTEM OVERLOAD (PURGE/NOPURGE)

Certain overload conditions, where all of a given system resource (for example, main storage) has been allocated and where each task requires still more of that resource, can occur in CICS/VS. The result is a situation in which no task is able to continue processing and no new task can be initiated; the system stalls.

CICS/VS has the capability to detect certain system stall conditions and take corrective action. Corrective action consists, in part, of purging (deleting) the lowest priority task in the system that is designated as stall purgeable.

A task is initially defined as purgeable or not purgeable in the program control table (PCT) entry associated with the transaction identification for that task. This entry is established by the system programmer at system generation. The application programmer can dynamically change the purgeability status of a task by issuing the

```
DFHKC TYPE=PURGE
```

macro instruction to indicate that the task is purgeable, or the

```
DFHKC TYPE=NOPURGE
```

macro instruction to indicate that the task is not purgeable. The designated status remains in effect for that task until another change is initiated or until the task is terminated. For example, a long-running task may issue a DFHKC TYPE=NOPURGE macro instruction prior to critical processing, then issue a DFHKC TYPE=PURGE macro instruction after that processing is completed. This ensures that the task is not stall-purged during the critical processing.

JOURNAL SERVICES

Journal management provides facilities for creating and managing special-purpose sequential data sets, called 'journals,' during real-time CICS/VS execution. Journals may contain any and all data the user needs, to facilitate subsequent reconstruction of events or data changes. For example, a journal might act as an audit trail, a change-file of data-base updates and additions, or a record of transactions passing through the system (often called a 'log').

In addition to the output services described in this section, journal management also provides support for:

- Operational control and disposition of volumes (see the CICS/VS Operations Guide [DOS/VS] or the CICS/VS Operations Guide [OS/VS])
- requests to switch volumes and/or read journal data sets during real-time CICS/VS execution (see the CICS/VS System Programmer's Reference Manual)

Requests for journal output services are made by issuing the journal control macro instruction (DFHJC), either directly from a user task or from a CICS/VS management program on behalf of a user task. Data may be directed to any journal data set specified in the journal control table (JCT), which defines the journals available during a particular CICS/VS execution. The JCT may define one or more journals on tape or direct access storage. Each journal is identified by a number known as the journal file identification. This number may range from 2 to 99; the value of 1 is reserved for a journal known as the system log.

All buffer space and other work areas needed for journal data set physical operations are acquired and managed by the journal control program (JCP). The user task supplies only the address and length of the data to be output. The data is moved to journal buffer space by JCP when building a journal record. The user task retains the use and control of the data and its CICS/VS storage area.

Journal output requests are serviced by JCP. Journal records are built into blocks compatible with standard variable-blocked format. JCP uses the host operating system's sequential access method to write the blocks to external storage devices.

Each logical journal record begins with the standard fullword length field, a user-specified identifier, and a system-supplied prefix. This data is followed in the journal record by any user-supplied prefix data (optional), and finally by the user-specified data. Journal control is designed so that the application programmer requesting output services need not be concerned further with the detailed layout and precise contents of journal records. He needs to know only which journal to use, what user data to specify, and what unique user-identifier to supply. Normally, he obtains this information from the application system analyst or the person(s) responsible for programs

for reading journal data sets. (See the CICS/VS System Programmer's Reference Manual.)

JCP builds journal records for output requests at the priority of the requesting program, under control of the TCA of the requesting program. However, the TCA is not used to communicate requests and to save/restore registers. Instead, a separate control area called a journal control area (JCA) is used; this area must be acquired by the task before any journal output requests are issued.

If no other event is in-process to the journal, output to a journal data set is also initiated under the requestor's TCA. However, output event completion is always processed under a different TCA--that of a high-priority journal task associated with the journal data set. Journal tasks are activated when CICS/VS execution begins, but are suspended when there are no output events outstanding. In a heavy load situation, where many user tasks request journal output while one output is in-process, a journal task initiates more output immediately after completion of the in-process output event.

The application programmer may specify parameter values for journal control requests in either of two ways:

- By including the parameters in operands of the DFHJC macro instruction by which journal services are requested, or
- By coding instructions that place the parameter values in fields of the JCA prior to issuing the DFHJC macro instruction

The second of these methods provides greater economy, in that the parameter values can be varied to meet the logic needs of the application, but only a single DFHJC macro instruction need be coded.

Journal output services that may be requested through the journal control macro instruction are introduced and explained in the following paragraphs. For detailed reference material, see "DFHJC Macro Instruction" in Chapter 7.

ACQUIRE THE JOURNAL CONTROL AREA (GETJCA)

If any journal output services are requested in an application program through DFHJC macro instructions, the application programmer must provide the symbolic definition of the journal control area (JCA) by copying the CICS/VS storage area map DFHJCADS. The JCA must be acquired for the task prior to any journal output requests by issuing the macro instruction:


```
DFHJC TYPE=GETJCA
```

The JCA may be acquired separately, as shown above, in which case no other operands are needed. Alternatively, the JCA may be acquired by and with the program's first journal output request; for example:

```
DFHJC TYPE=(GETJCA,PUT),                                        *
      .
      .
      .
```

If the latter approach is chosen, then it is not possible to place additional parameter values for the output request directly into the JCA prior to the request, because the JCA does not exist prior to this request. Macro warning messages are issued and the request is not processed if any such request is attempted.

In addition to acquiring the JCA for the task, the DFHJC TYPE=GETJCA macro instruction establishes addressability to the area by moving the contents of the JCA address field (TCAJCAAD) to JCABAR, the base locator specified for the area. Once acquired for the task, the JCA is reused for all subsequent journal requests issued by or on behalf of the task. Subsequent TYPE=GETJCA requests only cause JCABAR to be reloaded with the same value. The JCA may not be released by the user.

The following examples show how to acquire the journal control area (JCA) for the task:

For Assembler language:

```
       COPY  DFHTCADS              COPY TCA SYMBOLIC DEFINITIONS
       .
       .
       .
JCABAR EQU   10                    ASSIGN BASE REGISTER FOR JCA
       COPY  DFHJCADS              COPY JCA SYMBOLIC DEFINITIONS
       .
       .
       .
GETJCA DFHJC TYPE=GETJCA           REQUEST ACQUISITION OF THE JCA
       .
       .
       .
```

For ANS COBOL:

```
   02 JCABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                            NOTE DEFINE BASE LOCATOR FOR JCA.
      .
      .
01 DFHTCADS COPY DFHTCADS.         NOTE COPY TCA SYMBOLIC DEFINITIONS.
      .
      .
      .
01 DFHJCADS COPY DFHJCADS.         NOTE COPY JCA SYMBOLIC DEFINITIONS.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.        NOTE LOAD TCA BASE LOCATOR VALUE.
      .
      .
      .
GETJCA.
      DFHJC TYPE=GETJCA            REQUEST ACQUISITION OF THE JCA
```

For PL/I:

```
%INCLUDE DFHTCADS;                 /*COPY TCA SYMBOLIC DEFINITIONS*/
      .
      .
      .
%INCLUDE DFHJCADS;                 /*COPY JCA SYMBOLIC DEFINITIONS*/
      .
      .
      .
GETJCA:
      DFHJC TYPE=GETJCA            REQUEST ACQUISITION OF THE JCA
      .
      .
```

CREATE A JOURNAL RECORD AND WAIT FOR OUTPUT (PUT)

The application programmer may request journal output services by issuing the macro instruction:

```
DFHJC TYPE=PUT,                                                    *
      .
      .
      .
```

A journal record is created and written as output immediately. The requesting task waits until the output has been ccmpleted. By using this DFHJC macro instruction, the application programmer ensures that the journal record is written on the external storage device associated with the journal before processing continues; the task is said to be 'synchronized' with the output event. Most CICS/VS-provided data output service is performed in a synchronous manner.

The application programmer may request synchronous journal output services either by a DFHJC TYPE=PUT macro instruction as above, or by specifying DFHJC TYPE=(WRITE,WAIT). In both cases, certain additional keyword operands are mandatory. These keywords are JFILEID (the journal data set to receive data), JCDADDR (the address of the user data to be included in the journal record), JCDLGTH (the length of the user data), and JTYPEID (the two-byte user-specified hexadecimal identifier for the journal record). Optional accompanying keywords are PFXADDR (the address of user prefix data for inclusion in the journal record) and PFXLGTH (the length of the user prefix data); the application programmer may also include keyword operands to direct control to exception-handling routines in the program. (See "Test Response to a Request for Journal Services" in this chapter.)

The following examples show how to request and wait for journal output service.

For Assembler language:

```
        COPY  DFHTCADS              COPY TCA SYMBOLIC DEFINITIONS
        .
        .
        .
JCABAR  EQU   10                    ASSIGN BASE REGISTER FOR JCA
        COPY  DFHJCADS              COPY JCA SYMBOLIC DEFINITIONS
        .
        .
        .
FWACBAR EQU   9                     ASSIGN BASE REGISTER FOR FWA
        COPY  DFHFWADS              COPY FWA SYMBOLIC DEFINITIONS
RECORD  DS    0CL90
KEYDATA DS    0CL8
ACCNTNO DS    PL4
AMOUNT  DS    PL4
NAME    DS    CL20
ADDRESS DS    CL40
        .
        .
        .
        DFHJC TYPE=PUT,             REQUEST SYNCHRONOUS OUTPUT      *
              JFILEID=2,            TO JOURNAL ID 2,                *
              JCDADDR=KEYDATA,      OF THE 'KEY' DATA,              *
```

```
               JCDLGTH=8,           OF LENGTH=8 BYTES.                  *
               JTYPEID=0F01,        (IDENTIFIER FOR JOURNAL RECORD)     *
               NORESP=OK            BRANCH ADDR FOR GOOD RESPONSE
          .
          .
          .
OK        DS    0H
          .
          .
```

For ANS COBOL:

```
   02 JCABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                             NOTE DEFINE BASE LOCATOR FOR JCA.
      .
      .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY TCA SYMBOLIC DEFINITIONS.
      .
      .
      .
01 DFHJCADS COPY DFHJCADS.          NOTE COPY JCA SYMBOLIC DEFINITIONS.
      .
      .
      .
01 DFHFWADS COPY DFHFWADS.          NOTE COPY FWA SYMBOLIC DEFINITIONS.
   02 RECORD.
      03 KEYDATA PICTURE X(8).
      03 NAME PICTURE X(20).
      03 ADDRESS PICTURE X(20).
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.        NOTE LOAD TCA BASE LOCATOR VALUE.
      .
      .
      .
      DFHJC TYPE=PUT,              REQUEST SYNCHRONOUS OUTPUT          *
            JFILEID=2,             TO JOURNAL ID 2,                    *
            JCDADDR=KEYDATA,       OF THE 'KEY' DATA,                  *
            JCDLGTH=8,             OF LENGTH=8 BYTES.                  *
            JTYPEID=0F01,          (IDENTIFIER FOR JOURNAL RECORD)     *
            NORESP=OK              BRANCH ADDR FOR GOOD RESPONSE
      .
      .
      .
OK.
      .
      .
      .
```

For PL/I:

```
%INCLUDE DFHTCADS;                 /*COPY TCA SYMBOLIC DEFINITIONS*/
      .
      .
      .
%INCLUDE DFHJCADS;                 /*COPY JCA SYMBOLIC DEFINITIONS*/
      .
      .
      .
%INCLUDE DFHFWADS;                 /*COPY FWA SYMBOLIC DEFINITIONS*/
   02 RECORD,
```

```
      03 KEYDATA CHAR (8),
         04 ACCNTNO CHAR (4),
         04 AMOUNT CHAR (4),
      03 NAME CHAR (20),
      03 ADDRESS CHAR (40),
      .
      .
      .

      DFHJC TYPE=PUT,              REQUEST SYNCHRONOUS OUTPUT          *
            JFILEID=2,             TO JOURNAL ID 2,                    *
            JCDADDR=KEYDATA,       OF THE 'KEY' DATA,                  *
            JCDLGTH=8,             OF LENGTH=8 BYTES.                  *
            JTYPEID=0F01,          (IDENTIFIER FOR JOURNAL RECORD)     *
            NORESP=OK              BRANCH ADDR FOR GOOD RESPONSE
      .
      .
      .
OK:
      .
      .
      .
```

CREATE A JOURNAL RECORD FOR ASYNCHRONOUS OUTPUT (WRITE)

The application programmer may also request journal output services by issuing the macro instruction:

```
DFHJC TYPE=WRITE,                                                    *
      .
      .
      .
```

This request causes a journal record to be created in the journal buffer area, but allows the requesting task to retain control and thus to continue with other processing. The task may check and wait for output completion (that is, synchronize) at some later time. If that is intended, the requesting program should save the event control number (four bytes) returned in JCAECN after a successful TYPE=WRITE request.

Additional keyword operands applicable to TYPE=WRITE requests are as described above under "Create a Journal Record and Wait for Output," together with two further keywords, STARTIO and COND. To understand the use of these keywords, which apply only to asynchronous journal output requests, it is necessary to understand how journal control buffer management and output management are performed.

The basic process of building journal records in the buffer space of a given journal continues until such time as one of the following situations occurs:

- A request is made for synchronous output of a journal record.
- A request is rejected because of insufficient journal buffer space.
- The available buffer space is reduced below a user-specified level (see the CICS/VS System Programmer's Reference Manual).

At that time, all journal records present in the buffer--including any 'deferred' output resulting from asynchronous requests--are written to external storage, as one block.

If a task creates deferred output and delays synchronizing, the deferred output may be written 'for free' along with other requests;

when the task attempts to synchronize, there will be no need for it to wait. Thus, the advantages that may be gained by deferring journal output are: (1) transactions may get better response times by waiting less, (2) the load of physical I/O requests on the host system may be reduced, and (3) journal data sets may contain fewer but larger blocks and so better utilize external storage devices.

However, these advantages are achievable only at the cost of more buffer space and greater programming complexity. It is necessary to plan and program to control synchronizing with journal output. Additional decisions which depend on the data content of the journal record and how it is to be used must be made in the application program. In any case, the full benefit of deferring journal output is obtained only when the load on the journal data set is high.

The STARTIO keyword governs whether output is to be initiated (YES) or not (NO); in either case, control returns directly to the requesting program. The default option of NO should be used whenever possible because, if every journal request uses STARTIO=YES, no improvement over synchronous output requests, in terms of reducing the number of physical I/O operations and increasing the average block size, is possible.

The COND keyword governs what happens if the journal buffer space available at the time is not sufficient to contain the journal record for the request. If the default option COND=NO is taken, the requesting task loses control. The contents of the current buffer are written out, and the journal record for this request is built in the resulting freed buffer space before control returns to the requesting task.

If the requesting task is not willing to lose control--for example, if some housekeeping must be performed before other tasks get control--then COND=(YES,symbolic address) should be specified. If buffer space is momentarily insufficient, no journal record is built for the request, and control is returned directly to the requesting program at the location identified by symbolic address. The requesting program can perform any housekeeping needed before reissuing the journal output request.

The following example shows how to request deferred journal output, but ensure that the requesting task retains control to perform housekeeping, if necessary.

For Assembler language:

```
         COPY  DFHCSADS            COPY CSA SYMBOLIC DEFINITIONS
COMDATA  DS    CL10                AND COMMON WORK AREA
         .
         .
         .
         COPY  DFHTCADS            COPY TCA SYMBOLIC DEFINITIONS
SAVEDATA DS    CL10                SAVE AREA FOR COMMON DATA
MYDATA   DS    CL10                AREA FOR MY DATA
         .
         .
         .
JCABAR   EQU   10                  ASSIGN BASE REGISTER FOR JCA
         COPY  DFHJCADS            COPY JCA SYMBOLIC DEFINITIONS
         .
         .
         .
         MVC   SAVEDATA,COMDATA    SAVE COMMON DATA
         MVC   COMDATA,MYDATA      REPLACE BY MY DATA FOR WORKING
         .
         .
```

```
         .
         DFHJC TYPE=WRITE,          REQUEST ASYNCHRONOUS OUTPUT          *
               JCDADDR=COMDATA,     OF COMMON DATA AREA,                 *
               JCDLGTH=10,          LENGTH=10 BYTES,                     *
               JFILEID=SYSTEM,      TO SYSTEM LOG.                       *
               JTYPEID=0101,        (IDENTIFIER FOR JOURNAL RECORD)      *
               STARTIO=NO,          REQUEST DEFERRED OUTPUT,             *
               COND=(YES,RETRY),    BUT RETAIN CONTROL IF BUFFER FULL.   *
               NORESP=OK            BRANCH ADDR FOR GOOD RESPONSE
         .
         .
         .
OK       DS    0H
         .
         .
         .
RETRY    DS    0H                   HOUSEKEEPING:
         MVC   MYDATA,COMDATA       MOVE DATA, THEN
         MVC   COMDATA,SAVEDATA     RESTORE COMMON DATA.
         DFHJC TYPE=WRITE,          REQUEST ASYNCHRONOUS OUTPUT          *
               JCDADDR=MYDATA,      OF DATA,                             *
               JCDLGTH=10,          LENGTH=10 BYTES,                     *
               JFILEID=SYSTEM,      TO SYSTEM LOG.                       *
               JTYPEID=0101,        (IDENTIFIER FOR JOURNAL RECORD)      *
               COND=NO,             IF BUFFER FULL, WE'LL WAIT.          *
               STARTIO=NO,          DEFER OUTPUT.                        *
               NORESP=OK            BRANCH ADDR FOR GOOD RESPONSE
```

For ANS COBOL:

```
   02 JCABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                            NOTE DEFINE BASE LOCATOR FOR JCA.
      .
      .
01 DFHCSADS COPY DFHCSADS.         NOTE COPY CSA SYMBOLIC DEFINITIONS.
   02 COMDATA PICTURE X(10).       NOTE DEFINE COMMON DATA AREA.
      .
      .
      .
01 DFHTCADS COPY DFHTCADS.         NOTE COPY TCA SYMBOLIC DEFINITIONS.
   02 SAVEDATA PICTURE X(10).      NOTE SAVE AREA FOR COMMON DATA.
   02 MYDATA PICTURE X(10).        NOTE AREA FOR MY DATA.
      .
      .
      .
01 DFHJCADS COPY DFHJCADS.         NOTE COPY JCA SYMBOLIC DEFINITIONS.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.        NOTE LOAD TCA BASE LOCATOR VALUE.
      .
      .
      .
   MOVE COMDATA TO SAVEDATA.       NOTE SAVE COMMON DATA.
   MOVE MYDATA TO COMDATA.         NOTE REPLACE COMMON WITH MINE FOR WORK.
      .
      .
      .
      DFHJC TYPE=WRITE,            REQUEST ASYNCHRONOUS OUTPUT          *
            JCDADDR=COMDATA,       OF COMMON DATA AREA,                 *
            JCDLGTH=10,            LENGTH=10 BYTES,                     *
            JFILEID=SYSTEM,        TO SYSTEM LOG.                       *
            JTYPEID=0101,          (IDENTIFIER FOR JOURNAL RECORD)      *
```

```
            STARTIO=NO,             REQUEST DEFERRED OUTPUT,            *
            COND=(YES,RETRY),       BUT RETAIN CONTROL IF BUFFER FULL.  *
            NORESP=OK               BRANCH ADDR FOR GOOD RESPONSE
      .
      .
      .
OK.
      .
      .
RETRY.                              NOTE DO HOUSEKEEPING, THEN RETRY.
   MOVE COMDATA TO MYDATA.          NOTE MOVE DATA, THEN.
   MOVE SAVEDATA TO COMDATA.        NOTE RESTORE COMMON DATA.
      DFHJC TYPE=WRITE,             REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=MYDATA,         OF MY DATA,                         *
            JCDLGTH=10,             LENGTH=10 BYTES,                    *
            JFILEID=SYSTEM,         TO SYSTEM LOG.                      *
            JTYPEID=0101,           (IDENTIFIER FOR JOURNAL RECORD)     *
            STARTIO=NO,             REQUEST DEFERRED OUTPUT,            *
            COND=NO,                BUT IF BUFFER FULL, WE CAN WAIT.    *
            NORESP=OK               BRANCH ADDR FOR GOOD RESPONSE
```

For PL/I:

```
%INCLUDE DFHCSADS;                  /*COPY CSA SYMBOLIC DEFINITIONS*/
   02 COMDATA CHAR (10);            /*AND COMMON WORK AREA*/
      .
      .
      .
%INCLUDE DFHTCADS;                  /*COPY TCA SYMBOLIC DEFINITIONS*/
   02 SAVEDATA CHAR (10),           /*SAVE AREA FOR COMMON DATA*/
   02 MYDATA CHAR (10),             /*AREA FOR MY DATA*/
      .
      .
      .
%INCLUDE DFHJCADS;                  /*COPY JCA SYMBOLIC DEFINITIONS*/
      .
      .
      .
SAVEDATA=COMDATA;                   /*SAVE COMMON DATA*/
COMDATA=MYDATA;                     /*REPLACE BY MY DATA FOR WORKING*/
      .
      .
      .
      DFHJC TYPE=WRITE,             REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=COMDATA,        OF COMMON DATA AREA,                *
            JCDLGTH=10,             LENGTH=10 BYTES,                    *
            JFILEID=SYSTEM,         TO SYSTEM LOG.                      *
            JTYPEID=0101,           (IDENTIFIER FOR JOURNAL RECORD)     *
            STARTIO=NO,             REQUEST DEFERRED OUTPUT,            *
            COND=(YES,RETRY),       BUT RETAIN CONTROL IF BUFFER FULL.  *
            NORESP=OK               BRANCH ADDR FOR GOOD RESPONSE
      .
      .
      .
OK:
      .
      .
      .
RETRY:                              /*HOUSEKEEPING:*/
MYDATA=COMDATA;                     /*MOVE DATA, THEN*/
COMDATA=SAVEDATA;                   /*RESTORE COMMON DATA.*/
      DFHJC TYPE=WRITE,             REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=MYDATA,         OF MY DATA,                         *
            JCDLGTH=10,             LENGTH=10 BYTES,                    *
```

```
         JFILEID=SYSTEM,        TO SYSTEM LOG.                          *
         JTYPEID=0101,          (IDENTIFIER FOR JOURNAL RECORD)         *
         STARTIO=NO,            REQUEST DEFERRED OUTPUT,                *
         COND=NO,               BUT IF BUFFER FULL, WE CAN WAIT.        *
         NORESP=OK              BRANCH ADDR FOR GOOD RESPONSE
```

SYNCHRONIZE WITH THE OUTPUT OF A JOURNAL RECORD (WAIT)

If the application programmer creates a journal record for deferred (asynchronous) output, he may subsequently ensure that the record has been written (that is, synchronize) by issuing the macro instruction:

```
DFHJC TYPE=WAIT,                                                       *
      .
      .
      .
```

If the journal record output has already been completed (see discussion above under "Create a Journal Record for Asynchronous Output"), then control returns directly to the requesting task. If not, the output is initiated and the requesting task waits until the operation has been completed.

Before issuing a synchronizing request, the task must ensure that the event control number (four bytes) corresponding to the journal record in question is in field JCAECN of the JCA. An event control number is returned in JCAECN after every successful journal output request. Since the JCA is used for every journal request issued by the task (or by CICS/VS on its behalf), the requesting program must save the event control number immediately after an asynchronous output request if it is to be used later. This is necessary because the particular event control number may be overwritten during reuse of the JCA.

If the JCA is not reused between the output request and the synchronize request, the requesting program need not save and restore the event control number. It is the user's responsibility to determine whether or not he needs to save and restore it.

If the requesting program has made a succession of successful asynchronous output requests to the same journal data set, it is only necessary to synchronize on the last of these requests to ensure that all of the journal records have reached the external storage device. This may be done either by issuing a stand-alone DFHJC TYPE=WAIT request, or by making the last output request itself synchronous--a DFHJC TYPE=PUT or TYPE=(WRITE,WAIT).

The following examples show a typical sequence of instructions to request to synchronize with the output of a journal record.

For Assembler language:

```
         COPY  DFHTCADS          COPY TCA SYMBOLIC DEFINITIONS
SAVEDECN DS    CL4               SAVED EVENT CONTROL NUMBER
JDATA    DS    CL36              DATA TO WRITE TO JOURNAL
         .
         .
         .
JCABAR   EQU   10                ASSIGN BASE REGISTER FOR JCA
         COPY  DFHJCADS          COPY JCA SYMBOLIC DEFINITIONS
         .
         .
         .
```

```
         DFHJC TYPE=WRITE,          REQUEST ASYNCHRONOUS OUTPUT          *
               JCDADDR=JDATA,       OF DATA AT JDATA,                    *
               .                    ETC.                                 *
               .                                                         *
               .                                                         *
               NORESP=OK1           BRANCH TO OK1 IF GOOD RESPONSE
         .
         .
OK1      DS    0H
         MVC   SAVEDECN,JCAECN      SAVE EVENT CONTROL NUMBER
         .
         .
         .
         MVC   JCAECN,SAVEDECN      RESTORE EVENT CONTROL NUMBER,
         DFHJC TYPE=WAIT,           AND SYNCHRONIZE WITH OUTPUT.         *
               NORESP=OK2           BRANCH TO OK2 IF GOOD RESPONSE
         .
         .
         .
OK2      DS    0H
         .
         .
         .
```

For ANS COBOL:

```
   02 JCABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
                                    NOTE DEFINE BASE LOCATOR FOR JCA.
      .
      .
      .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY TCA SYMBOLIC DEFINITIONS.
   02 SAVEDECN PICTURE X(4).        NOTE SAVED EVENT CONTROL NUMBER.
   02 JDATA PICTURE X(36).          NOTE DATA TO WRITE TO JOURNAL.
      .
      .
      .
01 DFHJCADS COPY DFHJCADS.          NOTE COPY JCA SYMBOLIC DEFINITIONS.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.         NOTE LOAD TCA BASE LOCATOR VALUE.
      .
      .
      .
      DFHJC TYPE=WRITE,             REQUEST ASYNCHRONOUS OUTPUT          *
            JCDADDR=JDATA,          OF DATA AT JDATA,                    *
            .                       ETC.                                 *
            .                                                            *
            .                                                            *
            NORESP=OK1              BRANCH TO OK1 IF GOOD RESPONSE
      .
      .
      .
OK1.
   MOVE JCAECN TO SAVEDECN.         NOTE SAVE EVENT CONTROL NUMBER.
      .
      .
      .
   MOVE SAVEDECN TO JCAECN.         NOTE RESTORE EVENT CONTROL NUMBER.
      DFHJC TYPE=WAIT,              AND SYNCHRONIZE WITH OUTPUT.         *
            NORESP=OK2              BRANCH TO OK2 IF GOOD RESPONSE.
      .
```

```
      .
      .
OK2.
      .
      .
      .
```

For PL/I:

```
%INCLUDE DFHTCADS;                  //*COPY TCA SYMBOLIC DEFINITIONS*/
   02 SAVEDECN CHAR (4),            /*SAVED EVENT CONTROL NUMBER*/
   02 JDATA CHAR (36);              /*DATA TO WRITE TO JOURNAL*/
      .
      .
      .
%INCLUDE DFHJCADS;                  /*COPY JCA SYMBOLIC DEFINITIONS*/
      .
      .
      .
      DFHJC TYPE=WRITE,             REQUEST ASYNCHRONOUS OUTPUT          *
            JCDALDR=JDATA,          OF DATA AT JDATA,                    *
            .                       ETC.                                 *
            .                                                            *
            .                                                            *
            NORESP=OK1              BRANCH TO OK1 IF GOOD RESPONSE
      .
      .
      .
OK1:
SAVEDECN=JCAECN;                    /*SAVE EVENT CONTROL NUMBER*/
      .
      .
      .
JCAECN=SAVEDECN;                    /*RESTORE EVENT CONTROL NUMBER,*/
      DFHJC TYPE=WAIT,              AND SYNCHRONIZE WITH OUTPUT.         *
            NORESP=OK2              BRANCH TO OK2 IF GOOD RESPONSE
      .
      .
      .
OK2:
      .
      .
      .
```

TEST RESPONSE TO A REQUEST FOR JOURNAL SERVICES

Following processing of a request for journal services, a response code is provided in the JCA; this code indicates that the request was successfully executed or that one of several exception (or error) conditions occurred. The requesting program can check the response code and pass control to an appropriate exception-handling routine in the program. This may be accomplished in any of three ways:

1. Include the symbolic addresses of the program's exception-handling routines in keyword operands within the DFHJC journal output macro request.

2. Include the symbolic addresses in keyword operands within a

```
DFHJC TYPE=CHECK,                                                        *
      .
      .
      .
```

macro instruction immediately following the DFHJC journal output macro request.

3. Provide separate instructions to test the response code immediately following the DFHJC journal output macro request.

Under either of the first two methods above, the application programmer need not be concerned with the precise setting of the response code. He need only understand the keyword operands and be sure to provide for all conditions that can occur. Valid keywords are NORESP, INVREQ, IDERROR, LERROR, IOERROR, and NOTOPEN; these keywords are described in detail under "DFHJC Macro Instruction" in Chapter 7.

When the third method above is used, the application programmer must know the actual settings of the response code, which is returned at JCAJCRC. The possible response codes and the requests, conditions, and keyword operands to which they correspond are identified in Figure 6-10.

Journal Request by DFHJC Macro Instruction	Condition	Response Code		
		Assembler	ANS COBOL	PL/I
PUT, WRITE, WAIT, CHECK	NORESP (Normal Response)	X'ØØ'	12-0-1-8-9 (JCARCNR)	00000000
PUT,WRITE, WAIT, CHECK	IDERROR (Journal Identification Error)	X'Ø1'	12-1-9 (JCARCIDE)	00000001
PUT, WRITE, CHECK	LERROR (Journal Record Length Error)	X'Ø6'	12-6-9 (JCARCLE)	00000110
PUT, WAIT, CHECK	IOERROR (Output I/O Error)	X'Ø7'	12-7-9 (JCARCIOE)	00000111
PUT, WRITE WAIT, CHECK	NOTOPEN (Journal Not Open)	X'Ø5'	12-5-9 (JCARCNOE)	00000101
PUT, WRITE, WAIT, CHECK	INVREQ (Invalid Request)	X'Ø2'	12-2-9 (JCARCIRE)	00000010

NOTE: The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.

Figure 6-10. Journal Control Response Codes

If the application programmer does not provide for checking a particular response code and the corresponding condition occurs, program execution resumes at the instruction immediately following the DFHJC macro instruction which requested the journal service.

RECOVERY/RESTART SERVICES

Sync point management works in conjunction with other CICS/VS components, such as transient data management and file management, to provide the user with facilities needed for an emergency restart of CICS/VS after an abnormal termination of CICS/VS. In an emergency restart, changes made in protected resources (that is, in transient data intrapartition queues) can be backed out for tasks that were "in flight" at the time of failure. This backout is based upon information about the tasks recorded on a system log during execution.

Each synchronization point marks the completion of a logical unit of work. By definition, a logical unit of work (LUW) is an application programmer-defined unit of work that performs a complete processing function. One task may perform one LUW, or several LUWs, generally, delimited by conversational terminal operations (a terminal write, followed by a terminal read). The completion of a logical unit of work, or sync point, can be explicitly requested by the application program by means of a

```
DFHSP TYPE=USER
```

macro instruction. A sync point is always requested by CICS/VS at termination of a task.

The completion of a logical unit of work indicates to CICS/VS that:

- All updates or modifications performed by the task are logically complete, and should not be backed out if a system failure occurs.
- Functions requested prior to the synchronization point, but deferred until the end of the logical unit of work, are to be processed, even if a subsequent system failure occurs. An example of such an operation is a purge of a transient data intrapartition queue, as requested by the application program.
- All resources protected automatically on behalf of the task up to this point are to be released. An example of such a resource may be a transient data intrapartition destination that is logically associated with the task or a resource previously enqueued by the user.
- All resources previously enqueued by the user are dequeued.

The location of a sync point for a task on the system log data set, relative to other logged activity for that task, determines the extent to which CICS/VS (or user programs) may need to provide transaction backout. Generally, sync points are not needed for short-duration tasks.

RESTART RECOVERY

Sync points are also used by CICS/VS to delimit the extent to which user data set modifications may need to be backed out for a task. During emergency restart, CICS/VS collects all user data set modifications for tasks that were engaged in a LUW at the time of uncontrolled shutdown and copies them in a restart data set. The modifications can then be read by the CICS/VS transaction backout program or by user-written programs executed during the post-initialization phase of restart.

Through these facilities, sync point management not only permits emergency restart but also provides the means by which the activity required for such restart can be controlled by the user. The functions performed by other CICS/VS programs involved in sync point/uncontrolled shutdown/emergency restart activities are explained in greater detail

in the CICS/VS System Programmer's Reference Manual and CICS/VS System/Application Design Guide.

A sync point request for a task that is scheduled to use a DL/I resource implies the release of that resource. This means that if, after issuing a DFHSP TYPE=USER macro instruction, access to a DL/I data base is required, the desired PSB must be rescheduled through the DFHFC TYPE=(DL/I,PSB) macro instruction. The previous position of that data base has been lost.

CHAPTER 7. SYSTEM MANAGEMENT MACRO INSTRUCTIONS - GENERAL FORMATS

DFHTC

The general formats of CICS/VS macro instructions used to request supervisory and data management services are presented in this chapter. This material is designed primarily for reference purposes and is ordered to coincide with the tutorial information presented in the preceding chapter. Since a functional organization is used in presenting the tutorial information, the macro formats are also presented under functional headings. For each macro, the following are indicated:

- Available operands and their formats and meanings
- Whether operands are required or optional
- Defaults, if any
- Mutually exclusive or dependent operands, if any
- Special considerations, if any

The notation used within the boxes in this chapter is explained under "Coding Aids" in the first chapter of this manual.

DFHTC MACRO INSTRUCTION

The terminal control macro instruction (DFHTC) is used to request any of a wide variety of terminal services. Some operands may be applicable to any of numerous terminals supported by CICS/VS. Others are applicable to only specific devices. Meaningful combinations of DFHTC operands for the various terminals (or for the logical units that are the VTAM equivalent of physical terminals) are provided in this section along with explanations.

Note: GET, PUT, PAGE, and CONVERSE are additional keyword parameters provided for coding convenience; they are combinations of other parameters and may be substituted where the others appear as follows:

- GET - same as READ, WAIT
- PUT - same as WRITE, WAIT
- CONVERSE - same as WRITE, READ, WAIT
- PAGE - same as ERASE, WRITE, READ, WAIT for 3270 or for 3270 in 2260 compatibility mode

This section includes discussions of the CICS/VS macro instruction used to communicate with VTAM-supported terminals. Refer to the CICS/VS Advanced Communication Guide for additional information on using CICS/VS to communicate with these devices.

INPUT OPERATIONS

TCAM-Supported Terminals (CICS/OS/VS Only)

Input operations for terminals supported by TCAM use the same operands applicable to the terminals for other access methods. With

exception of READB for the 3270, all operands applicable for input operations are supported by CICS/VS-TCAM support.

VTAM-Supported Logical Units

	DFHTC	TYPE=(READ[,WAIT][,SAVE]) [,IOTYPE={IMMED}] [{DELAY}] [,EODS=symbclic address]

TYPE=
describes the logical unit operations required.

READ
indicates that data is to be read from the logical unit.

WAIT
ensures that the logical unit operation requested in this macro instruction is completed before subsequent processing is carried out. WAIT can be coded separately from a READ (with IOTYPE=IMMED) to accomplish overlapping of logical unit I/O operations.

Note: When ANS COBOL is used, a WAIT must be included with every READ in a program.

SAVE
indicates that the TIOA used in a previous logical unit operation is not to be used as an input area for the read operation. A new TIOA is to be acquired.

IOTYPE=
specifies when the read operation is to be started.

IMMED
causes the read operation to be started immediately (as soon as the request is given to the logical unit control management module). This parameter enables overlapping of logical unit I/O operations with application program processing by coding a WAIT separately from the READ.

DELAY
causes the read operation to be started only when a WAIT is encountered. If this operand is omitted, IOTYPE defaults to the transaction option specified by TIOTYPE in the DFHPCT TYPE=ENTRY macro instruction. A task using deferred write or message protection always defaults to DELAY regardless of the PCT specification.

EODS=symbolic address
indicates the label of a user-written routine to receive control if end-of-data-set is received from a 3650 interpreter logical unit. The TIOA contains the EODS indicators.

This parameter applies only to 3650 interpreter logical units.

System/7

	DFHTC	TYPE=(READ[,WAIT][,SAVE][,{TRANSPARENT / PSEUDOBIN}])

TYPE=
describes the terminal operations required.

READ
indicates that data is to be read from a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every READ in a program.

SAVE
indicates that the TIOA used in a previous terminal operation is not to be used as an input area for the read operation. A new TIOA is to be acquired.

TRANSPARENT
indicates that the data being read is not to be translated.

PSEUDOBIN
indicates that the data being read is tc be translated from System/7 pseudobinary representation to hexadecimal. (For further informatic n concerning System/7 programming considerations, see "System/7 Considerations" in Chapter 11.)

2260 Display Station

	DFHTC	TYPE=({READ / READL}[,WAIT][,SAVE]

TYPE=
describes the terminal operations required.

READ
indicates that data is to be read from a terminal.

READL
indicates that the keyboard is to remain locked at the completion of a data transfer. This parameter is applicable only to CICS/OS/VS, but may be used on a CICS/DOS/VS application if compatibility with CICS/OS/VS is desired.

WAIT
ensures that the terminal operation requested in this macro is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every READ and READL in the program.

SAVE
indicates that the TIOA used in a previous terminal operation is not to be used as an input area for the read operation. A new TIOA is to be acquired.

2741 Communication Terminal

	DFHTC	TYPE=(READ[,WAIT]) ,RDATT=symbclic address

TYPE=
describes the terminal operations required.

READ
indicates that data is to be read from a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every READ in a program.

RDATT=symbolic address
is the symbolic address to which control is transferred if a read operation in response to this DFHTC TYPE=READ macro instruction is terminated by pressing of the Attention (ATTN) key rather than the Return key.

Note: This operand is meaningful only if 2741 Read Attention support has been generated in the CICS/VS system. See "2741 Read Attention and Write Break Support" in Chapter 11.

3270 Information Display System

	DFHTC	TYPE=({READ / READL / READB})[,WAIT][,SAVE][,TEXT]

TYPE=
describes the terminal operation required.

READ
indicates that data is to be read from a terminal.

READL
indicates that the keyboard is to remain locked at the completion of data transfer. This parameter is meaningful only for a 3270 operating in 2260 compatibility mode and is applicable only to CICS/OS/VS. However, the parameter may be used in a CICS/DOS/VS application program if upward compatibility with CICS/OS/VS is a consideration.

READB
reads the contents of the 3270 buffer, beginning at buffer location 0 and continuing until all contents of the buffer have been read. All character and attribute sequences (including nulls) appear in the input data stream in the same order that they appear in the 3270 buffer. READB cannot be specified for TCAM-supported terminals.

Note: Because of the relatively long transmission times required to transmit the entire contents of a remote 3270 buffer, the READB parameter should be used primarily for test and diagnostic purposes; the COPY parameter, which permits a selective transfer of buffer contents (see "Output Operations" later in this chapter), should be used where possible in all other cases.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every READ, READL, or READB in a program.

SAVE
indicates that the TIOA used in a previous terminal operation is not to be used as an input area for the read operation. A new TIOA is to be acquired.

TEXT
is meaningful only when used in conjunction with a READ request and specifies a temporary override of the uppercase translation feature of CICS/VS to allow this task to receive a message containing both uppercase and lowercase data.

3735 Programmable Buffered Terminal

	DFHTC	TYPE=(READ[,WAIT][,SAVE]) [,EOF=symbolic address]

TYPE=
describes the terminal operations required.

READ
indicates that data is to be read from a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every READ in a program.

SAVE
indicates that the TIOA used in a previous terminal operation is not to be used as an input area for the read operation. A new TIOA is to be acquired.

EOF=symbolic address
is the symbolic address of the routine to receive control when end of file is encountered on batch input. This operand can be used in a special initialization macro instruction, DFHTC EOF=symbolic address, to test for the end-of-file condition upon initial connection to a 3735. It must be included in the initialization section of the '3735' transaction (that is, of the application program to handle 3735 input), preceding other DFHTC macro instructions.

Note: When the EOF condition occurs, TIOATDL is set to binary zeros to indicate that the TIOA for the input operation contains no valid data.

3740 Data Entry System

	DFHTC	TYPE=(READ[,WAIT][,SAVE]) [,ENDFILE=symbolic address] [,ENDINPT=symbolic address]

TYPE=
describes the terminal operations required.

READ
indicates that data is to be read from a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every READ in a program.

SAVE
indicates that the TIOA used in a previous terminal operation is not to be used as an input area for the read operation. A new TIOA is to be acquired.

ENDFILE=symbolic address
is the symbolic address of the routine to receive control when end-of-file is encountered.

ENDINPT=symbolic address
is the symbolic address of the routine to receive control when end-of-input is reached.

All Other CICS/VS-Supported Terminals

	DFHTC	TYPE=(READ[,WAIT][,SAVE])

TYPE=
describes the terminal operations required.

READ
indicates that data is to be read from a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every READ in a program.

SAVE
indicates that the TIOA used in a previous terminal operation is not to be used as an input area for the read operation. A new TIOA is to be acquired.

OUTPUT OPERATIONS

TCAM-Supported Terminals (CICS/OS/VS Only)

	DFHTC	TYPE=(WRITE[,other optional parameters]) [,DEST={symbolic address / YES}] [,terminal dependent operands]

TYPE=
describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

other optimal parameters
indicates that other parameters may be coded by the user for the specific terminal being written to. Any parameter applicable to the terminal for other access methods may also be coded for TCAM operations. For example, LAST could be coded for a VTAM terminal or TRANSPARENT for System/3.

DEST=
indicates that the output message is to be sent to a TCAM destination other than the source TCAM terminal. This operand is meaningful only for terminals for which DEVICE=TCAM has been specified in the DFHTCT TYPE=SDSCI macro instruction.

symbolic name
is the symbolic name of the TCAM destination to which the message is to be sent.

YES
indicates that the application program has placed the four-byte message destination in TCTTEDES before issuing the WRITE. This method allows dynamic selection of the message destination.

terminal dependent operands
indicates that other operands applicable to the specific terminals for other access methods may also be coded for TCAM operations. For example, IOTYPE for VTAM terminals, WRBRK for the 2741, or LINEADR for the 3270.

VTAM-Supported Logical Units

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE][,LAST]) [,IOTYPE={IMMED / DELAY}] [,LDC={mnemonic / YES}] [,FMH={NO / YES}]

TYPE=

describes the logical unit operations required.

WRITE
indicates that data is to be written to a logical unit.

WAIT
ensures that the logical unit operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation.

LAST
signals CICS/VS that this WRITE is the last output for a transaction and, therefore, the end of a bracket operation.

IOTYPE=

specifies when the write operation is to be started.

IMMED
causes the write operation to be started immediately (as soon as the request is given to the logical unit control program). This parameter enables overlapping of logical unit I/O operations with application program processing by coding a WAIT separately from the WRITE.

DELAY
causes the write operation to be started only when a WAIT is encountered.

If this operand is omitted, IOTYPE defaults to the transaction option specified by TIOTYPE in the DFHPCT TYPE=ENTRY macro instruction. A task using deferred write or message protection always defaults to DELAY regardless of the PCT specification.

LDC=

specifies the mnemonic to be used by CICS/VS to determine the logical device code (LDC) that is to be transmitted in the function management header to the logical unit. This operand is used only for the 3601 logical unit (but not for the 3614 even if attached to the 3601).

mnemonic
is the two-character mnemonic used to determine the appropriate LDC numeric value. The mnemonic represents a LDC entry in the DFHTCT TYPE=LDC macro instruction.

YES
indicates that the application program has placed the mnemonic in TCATPIDM.

FMH=

indicates whether the function management header (FMH) has been placed in the TIOA by the application program. If this operand is omitted, NO is assumed. This operand is used only for the 3601 logical unit (but not for the 3614 even if attached to the 3601), 3650 host conversational (3270) logical units, and 3790 logical units.

For the 3601 and 3790 logical units, an FMH is required and is provided as described below. For the 3650 host conversational (3270) logical unit, the FMH is required if outboard maps are to be used; the FMH in such cases can be provided by BMS, if BMS is being used, or otherwise, by the application program.

NO
indicates that the application program has not placed the FMH in the TIOA. For the 3601 and 3790 logical units, CICS/VS is responsible for placing the FMH in the TIOA; if NO is specified, space must be reserved in the TIOA for the FMH. For the 3650 host conversational (3270) logical unit, CICS/VS does not build an FMH, and the data is trnasmitted unmodified to the 3651 by CICS/VS. Refer to the CICS/VS Advanced Communication Guide for size and format of the FMH for a specific terminal.

YES
indicates that the application program has placed the FMH into the TIOA.

System/3, System/370, 2770, 2780, or 3780

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE][,TRANSPARENT])

TYPE=
describes the terminal operation required.

WRITE
indicates that data is to be written to a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE except the final WRITE in a program.

SAVE
indicates that the TIOA used in the terminal operation is not to be released upon completion of the write operation.

TRANSPARENT
indicates that output is to be sent in transparent mode (with no recognition of control characters and accepting any of the 256 possible combinations of eight bits as valid transmittable data).

System/7

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE][,{TRANSPARENT / PSEUDOBIN}])

TYPE=

describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation.

TRANSPARENT
indicates that the data being written is not to be translated.

PSEUDOBIN
indicates that the data being written is to be translated from hexadecimal to System/7 pseudobinary representation. (For further information concerning System/7 programming considerations, see "System/7 Considerations" in Chapter 11.)

2260 Display Station

	DFHTC	TYPE=({WRITE / WRITEL}[,WAIT][,SAVE][,ERASE]) [,LINEADDR={number / YES}]

TYPE=

describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

WRITEL
indicates that the keyboard is to remain locked if locked previously, or to remain unlocked if unlocked previously, at the completion of data transfer. If DFHTC macro instructions are issued in the following sequence, the keyboard is locked or unlocked as indicated:

READ	L
WRITEL	L
READL	L
READL	L
WRITEL	L
WRITEL	L
WRITE	U
WRITEL	U
WRITEL	U
READL	L
WRITE	U

```
READL       L
READ        L
WRITEL      L
```

WAIT

ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a wait must be included with every WRITE and WRITEL except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon ccmpletion of the write operation.

ERASE
is meaningful only when used in conjunction with either WRITE or WRITEL and causes the screen tc be erased and the cursor returned to the upper left corner of the screen before writing occurs.

Note: To simply erase the screen, (1) place the address of a TIOA into TCTTEDA, (2) place a data length of 1 in TIOATDL, and (3) issue a DFHTC TYPE=(WRITE,ERASE) macro instruction. The TIOA should contain only a start symbol.

LINEADDR=
specifies that writing is to begin on a specific line of the 2260 screen.

number
is the hexadecimal equivalent of the starting line number. X'F0' through X'FB' correspond with the line numbers 1 through 12 respectively.

YES
indicates that the hexadecimal equivalent of the line number has been placed in TIOALAC.

2741 Communication Terminal

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE]) ,WRBRK=symbolic address

TYPE=
describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation.

WRBRK=symbolic address
is the symbolic address to which control is transferred if a write operation started in response to this DFHTC TYPE=WRITE macro instruction is interrupted by the terminal operator pressing the Attention (ATTN) key.

Note: This operand is meaningful only if 2741 Write Break support has been generated into the system, an option available

only under CICS/OS/VS. See "2741 Read Attention and Write Break Support."

2980 General Banking Terminal

	DFHTC	TYPE= CBUFF PASSBK

TYPE=
describes the terminal operations required.

CBUFF
is a stand-alone parameter used to place a message in the common buffer of the 2972 terminal control Unit; the 2972 associated with the current TCTTE receives the output message. Both write and wait are implied.

Note: The output message is translated according to the model of 2980 described by the current TCTTE. If more than one model is attached to a 2972 Terminal Control Unit, the contents of the common buffer are intelligible only to the model for which the message was translated. Since shift characters are added to the message by CICS/VS during translation, the length of the message is dependent upon the contents of the message. Up to 23 characters, including shift characters, can be transmitted.

PASSBK
is a stand-alone parameter used to cause output to be printed on a banking passbook. Both WRITE and WAIT are implied. If a passbook is not present, no printing occurs. An error message can be sent to the operator of the terminal associated with the requesting task.

3270 Information Display System

	DFHTC	TYPE=({WRITE / WRITEL / COPY / PRINT / ERASEAUP}[,WAIT][,SAVE][,ERASE]) [,LINEADR={number / YES}] [,CTLCHAR={hexadecimal number / YES}]

TYPE=
describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

WRITEL
indicates that the keyboard is to remain locked if locked previously, or to remain unlocked if unlocked previously,

at the completion of data transfer. This operand is meaningful only for a 3270 operating in 2260 compatibility mode.

If DFHTC macro instructions are issued in the following sequence, the keyboard is locked or unlocked as indicated:

```
READ        L
WRITEL      L
READL       L
READL       L
WRITEL      L
WRITEL      L
WRITE       U
WRITEL      U
WRITEL      U
READ        L
WRITE       U
READL       L
READ        L
WRITEL      L
```

COPY
is used to copy the format and data contained in the buffer of another terminal attached to the same 3271 control unit into a buffer of this terminal. The terminal from which data is to be copied can be identified in either of two ways:

- TIOATDL is set to a value of 1 and the first byte of the output data area (TIOADBA) is set to the physical address of the terminal to be copied; or
- TIOATDL is set to a value of 4 and the first four bytes of the output data area (TIOADBA) are set to the terminal identification of the terminal to be copied. If the terminal identification is less than four bytes in length, it must be left-justified with blank padding on the right.

The copy control character (CCC), which controls and defines the copy function to be performed, must be supplied in the CTLCHAR operand of this DFHTC macro instruction. Neither WRITE, ERASE, nor ERASEAUP can be specified in a DFHTC macro instruction that includes the COPY parameter.

PRINT
specifies that the data currently displayed on a 3277 or 3275 is to be printed on an eligible 3284 or 3286 printer.

ERASEAUP
is used to issue an "erase all unprotected" command and causes the following functions to be performed:

1. All unprotected fields are cleared to nulls (X'00').
2. The modified data tags (MDTs) in each unprotected field are reset to zero.
3. The cursor is positioned to the first unprotected field.
4. The keyboard is restored.

Neither WRITE, ERASE, nor COPY can be specified in a DFHTC macro instruction that includes the ERASEAUP parameter. No data stream is supplied. This operand is not meaningful for a 3270 operating in 2260 compatibility mode.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE, WRITEL, COPY, and ERASEAUP except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation.

ERASE
is meaningful only when used in conjunction with either WRITE or WRITEL and causes the screen to be erased and the cursor returned to the upper left corner of the screen before writing occurs.

Note: To simply erase the screen, (1) place the address of a TIOA into TCTTEDA, (2) place a data length of 0 into TIOATDL, and (3) issue a DFHTC TYPE=(WRITE, ERASE) macro instruction. If operating in 2260 compatibility mode, the TIOA should contain only a start symbol and the data length in TIOATDL should be set to 1 before issuing the DFHTC TYPE=(WRITE,ERASE).

LINEADR=
specifies that writing is to begin on a specific line of a 2260/2265 screen simulated on a 3270 operating in 2260 compatibility mode.

number
is the hexadecimal equivalent of the starting line number. For the 2260, X'F0' through X'FB' correspond with line numbers 1 through 12 respectively. For the 2265, X'F0' through X'FE' correspond with line numbers 1 through 15 respectively.

YES
indicates that the hexadecimal equivalent of the line number has been placed in TIOALAC.

CTLCHAR=
is used (1) in a DFHTC TYPE=WRITE macro instruction to provide the hexadecimal representation of the write control character (WCC) that controls the requested write operation, or (2) in a DFHTC TYPE=COPY macro instruction to provide the hexadecimal representation of the copy control character (CCC) that controls and defines the copy function to be performed.

hexadecimal number
is the hexadecimal representation of the WCC or CCC required for the operation specified in the TYPE= operand of this DFHTC macro instruction.

YES
indicates that the appropriate bit configuration has been placed in TIOACLCR.

For DFHTC TYPE=WRITE, if only the functions defined by the WCC are to be performed (that is, no data stream is to be supplied), TIOATDL must contain zero. If the CTLCHAR operand is omitted, all modified data tags are reset to zero, and the keyboard is restored. For DFHTC TYPE=COPY, if the CTLCHAR operand is omitted, the contents of the entire buffer (including nulls) are copied and the start printer flag is not on.

3735 Programable Buffered Terminal

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE][,NOTRANSLATE])

TYPE=
describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation.

NOTRANSLATE
prevents translation of form description program (FDP) records which are to be transmitted to a 3735 using ASCII transmission code. (For further information, see "3735 Considerations" in Chapter 11.)

3740 Data Entry System

	DFHTC	TYPE=([WRITE][,WAIT][,SAVE][,ENDFILE][,ENDOUTPUT] [,TRANSPARENT])

TYPE=
describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation.

ENDFILE
indicates that an end-of-file record is to be written to the terminal.

ENDOUTPUT
indicates that an end-of-output record is to be written to the terminal.

TRANSPARENT
indicates that output is to be sent in transparent mode (with no recognition of control characters and accepting any of the 256 possible combinations of eight bits as valid transmittable data).

All Other CICS/VS-Supported Terminals

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE])

TYPE=
describes the terminal operations required.

WRITE
indicates that data is to be written to a terminal.

WAIT
ensures that the terminal operation requested in this macro instruction is completed before subsequent processing is carried out.

Note: When ANS COBOL is used, a WAIT must be included with every WRITE except the final WRITE in a program.

SAVE
indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation.

MISCELLANEOUS OPERATIONS

Line Control

	DFHTC	TYPE={RESET / DISCONNECT}

TYPE=
describes the operation to be performed.

RESET
applies only to binary synchronous devices and is used to relinquish use of a communication line; the next BTAM type

of operation will be a read or write initial. RESET is not supported by CICS/VS-TCAM support.

DISCONNECT
applies only to switched lines and is used to break the line connection between the terminal and the computer; if the terminal is a buffered device, the data in the buffer(s) is lost. CICS/VS does not automatically disconnect a 3275 at the end of a transaction. A disconnection occurs at the request of a terminal operator, at the request of the application program (through this macro instruction), or after a specified number of time-outs are encountered by DFHTEP for the terminal. (Refer to the CICS/VS System Programmer's Reference Manual for information on DFHTEP.)

3650 Program Request

	DFHTC	TYPE=PROGRAM ,PRGNAME=name [,VALID=address] [,NONVAL=address] [,CONNECT={ACTIVATE / CONVERSE}] [,NORESP=address]

The DFHTC TYPE=PROGRAM macro instruction is specified only when a connection is being initiated from the host CPU with a 3650 application program. Refer to the CICS/VS Advanced Communication Guide for details on communicating with a 3650 application program.

TYPE=PROGRAM
is used to request the loading of a 3650 application program. If the program is loaded, control is returned to the next sequential instruction following the DFHTC TYPE=PROGRAM macro instruction unless NORESP=program is specified. Otherwise, control is returned to an address specified by one of the other operands of the macro instruction as listed below.

PRGNAME=name
indicates the name the 3650 application program. This name information (up to eight characters) is transmitted to the 3651 for verification by the 3650 control program.

VALID=address
indicates the label of a user-coded routine to receive control if the name specified in the PRGNAME operand is valid but sufficient resources are not available in the 3651 to initiate the 3650 application program. This routine can determine whether a DFHKC TYPE=SCHEDULE macro instruction is to be issued in order to restart the 3650 application program later.

NONVAL=address
indicates the label of a user-coded routine to receive control if the name specified in the PRGNAME operand is invalid.

CONNECT
specifies the type of connection to be established.

ACTIVATE
specifies that the 3650 application program will not communicate with the host CPU.

CONVERSE
specifies that the 3650 application program will communicate with the host CPU.

NORESP=address
indicates the label of a user-coded routine to receive control if there is a no error response.

EODS for 3650 Interpreter Logical Units

	DFHTC	TYPE=EODS

TYPE=EODS
causes an end of data set function management header (FMH) to be sent on behalf of the task. An I/O area need not be supplied by the CICS/VS application programmer.

Refer to the CICS/VS Advanced Communication Guide for details on communicating with a 3650 application program.

DFHFC MACRO INSTRUCTION

The file control macro instruction (DFHFC) is used to request file services as explained below.

RANDOMLY RETRIEVE DATA FROM A DATA SET

The general format of the DFHFC macro instruction to randomly retrieve data from a data set is as follows:

```
DFHFC  TYPE=GET
       [,DATASET=symbolic name]
       [,RDIDADR=symbolic address]
       [,SEGSET={symbolic name}]
                {YES         }
                {ALL         }
       [,INDEX={symbolic name}]
               {YES         }
       [,TYPOPER=UPDATE]
       [,RETMETH={RELREC}]  <-- DAM
                 {KEY   }
       [,ARGTYP={KEY}]      }
                {RBA}       }
       [,SRCHTYP={FKEQ}]    }
                 {FKGE}     } VSAM
                 {GKEQ}     }
                 {GKGE}     }
       [,MODE={MOVE  }]     }
              {LOCATE}      }
       [,NORESP=symbolic address]
       [,ERROR=symbolic address]
       [,DSIDER=symbolic address]
       [,SEGIDER=symbolic address]
       [,NOTFND=symbolic address]
       [,INVREQ=symbolic address]
       [,IOERROR=symbolic address]
       [,DUPDS=symbolic address]
       [,NOTOPEN=symbolic address]
       [,ILLOGIC=symbolic address]  <-- VSAM
```

where:

TYPE=GET
: indicates that retrieval of a record is required.

DATASET=symbolic name
: is the symbolic name of the primary data set to be accessed. When indirect accessing is involved, this is the name of the target data set (see "Indirect Accessing" in Chapter 11). The name must appear in the file control table (FCT). If this operand is omitted, the symbolic name is assumed to be in TCAFCDI.

RDIDADR=symbolic address
: is the symbolic address of the user's record identification field that contains the key of the record to be retrieved (for ISAM), the block reference field (for DAM), or the key or relative byte address (for VSAM). If this operand is omitted, the symbolic address is assumed to be in TCAFCRI. (For further details, see "record identification Field" in Chapter 11.)

SEGSET=
: indicates that the data set specified in the DATASET operand contains segmented records and identifies the segment set to be retrieved.

symbolic name
is the symbolic name of the segment set to be retrieved. The name must have been defined in the associated segment control section of the FCT.

YES
indicates that the symbolic name of the segment set to be retrieved has been placed in TCAFCSI.

ALL
is used when reading a segmented record to indicate that the entire logical record in an unpacked and aligned format is desired. SEGSET=ALL is assumed by CICS/VS when updating a segmented record; the entire logical record is unpacked and returned to the application program.

If this operand is omitted, and the DFHFC TYPE=GET macro instruction refers to a data set containing segmented records, the logical record is returned in its packed unaligned format.

INDEX=
indicates that indirect accessing is to be used and specifies the symbolic name of the highest level index data set to be used. (This index data set is the first data set accessed in the hierarchy.)

symbolic name
is the symbolic name of the highest level index data set to be accessed. The name must have been defined in the FCT.

YES
indicates that the symbolic name of the highest level index data set has been placed in TCAFCAI.

If the data set identified by this operand is a DAM data set, it cannot be blocked.

TYPOPER=UPDATE
indicates that a record is to be obtained for updating, or, if a VSAM key-sequenced data set is referred to, for either updating or deletion. If records in a protected VSAM keyed-sequential data set are to be updated or deleted, ARGTYP=KEY must be specified and SRCHTYP must be FKEQ. If the record is from a blocked DAM data set, the RETMETH operand must be specified. If TYPOPER=UPDATE is omitted, a read-only operation is assumed.

DAM Data Set

RETMETH=
applies only to blocked DAM data sets and is used to specify the argument type (retrieval method) for deblocking the data sets.

RELREC
specifies that retrieval is to occur by relative record, with the first to be record in a block considered to be record zero.

KEY
specifies that retrieval is to occur by key.

If TYPOPER=UPDATE is specified for a DAM data set, this operand is required. If this operand is omitted and a request to read a blocked DAM data set is issued, the entire physical record (block) is returned in the FIOA to the application program. The user's block reference field, required by DAM, contains the criteria for deblocking the data set. If a retrieved record is "undefined," the application program must determine the length of the record.

VSAM Data Set

ARGTYP=
describes the contents of the record identification field of a VSAM data set.

KEY
indicates that the record identification field contains a search key.

RBA
indicates that the record identification field contains a relative byte address.

SRCHTYP=
specifies how the search key in the record identification field is to be used when locating VSAM records. This operand is meaningful only when ARGTYP=KEY is specified or implied by default.

FKEQ
indicates that the search key is a full key and that only a record with an equal key satisfies the search.

FKGE
indicates that the search key is a full key and that the first record with a key equal to or greater than the search key satisfies the search.

GKEQ
indicates that the search key is a generic (partial) key, the binary length of which is specified in the first byte of the record identification field. The search is satisfied when a record whose key is equal to the search key (compared on only the number of bytes specified in the first byte of the record identification field) is found.

GKGE
indicates that the search key is a generic key and that the first record with a key equal to or greater than the search key (compared on only the number of bytes specified in the first byte of the record identification field) satisfies the search.

MODE=
is used to specify the processing mode for a read-only request to a VSAM data set.

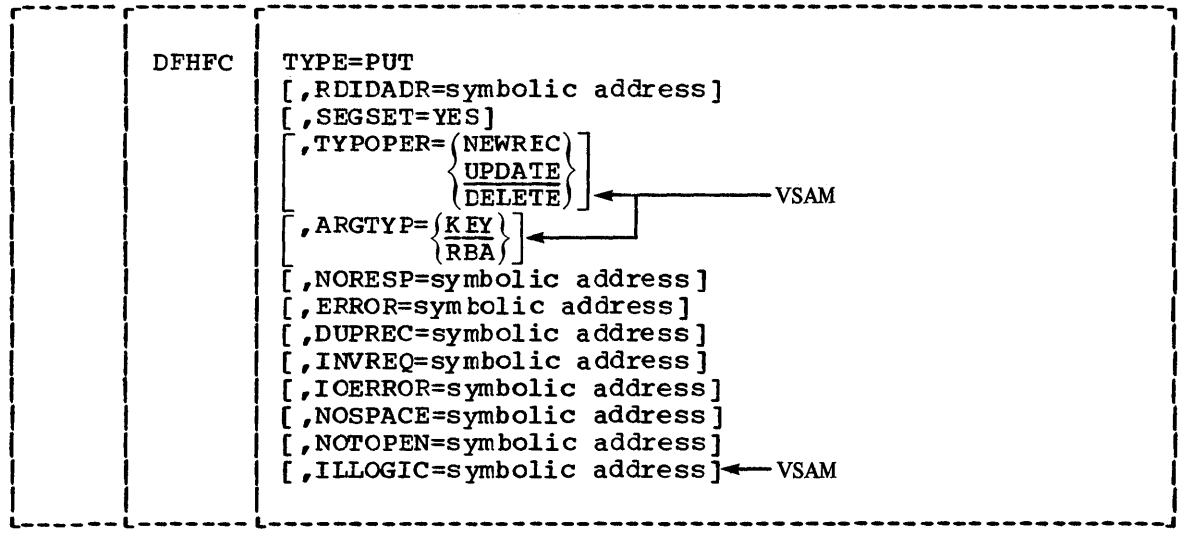
MOVE
specifies move mode processing. Upon return to the application program, TCAFCAA contains the address of the FIOA or FWA acquired for the read-only operation. If the data set referred to contains variable-length records, the traditional LLØØ length field is included as part of the logical record.

LOCATE
specifies locate mode processing. Upon return to the application program, TCAFCAA contains the address of a VSWA. The address of the retrieved record is at VSWAREA. If the data set referred to contains variable-length records, the traditional LLØØ length field is not retrieved as part of the logical record; instead, the length of the record is placed in VSWALEN. This parameter cannot be specified, if TYPOPER=UPDATE is specified and/or segmented records are being retrieved.

NORESP, ERROR, DSIDER, SEGIDER, NOTFND, INVREQ, IOERROR, DUPDS, NOTOPEN, and ILLOGIC (VSAM only)
are used to test the CICS/VS response to this request for record retrieval. These operands can be specified in this macro instruction or in a DFHFC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for File Services" in this chapter.

RANDOMLY UPDATE OR ADD DATA TO A DATA SET

The general format of the DFHFC macro instruction to randomly update or add data to a data set is as follows:



where:

TYPE=PUT
indicates that updating of a record or addition of a record is required.

Note: For records written to a variable length VSAM data set, the length of the record should be placed in an 'LL' field in the beginning of the record. This field is used by CICS to determine the length of the record and is not written to the data set.

RDIDADR=
is the symbolic address of the user's record identification field that contains the key (as required for ISAM), the block reference field (for DAM), or the key or relative byte address (for VSAM) of the record to be written. If this operand is omitted, the symbolic address is assumed to be in TCAFCRI. This field is used when adding a new record or when updating an existing record in a nonkeyed DAM data set without previously reading it for update. Note that this operand must not refer to a field in the FWA because the FWA might be freed before the write occurs.

Note: The DFHFC TYPE=PUT,TYPOPER=NEWREC macro instructions of a VSAM mass-insert operation may specify the same record identification field or different record identification fields.

SEGSET=YES
indicates that the data set to be added to or updated contains segmented records. If this operand is omitted, file control does not perform its normal packing operation on segmented records (see "Segmented Records" in Chapter 11).

TYPOPER=
describes the file operation to be performed.

NEWREC
indicates that a new record is to be added to an existing data set.

UPDATE
indicates that an existing record retrieved previously by a DFHFC TYPE=GET,TYPOPER=UPDATE instruction is to be updated (in effect, rewritten to the data set).

DELETE
is valid only when a VSAM key-sequenced data set is being accessed and indicates that a record previously retrieved for update by a DFHFC TYPE=GET,TYPOPER=UPDATE request is to be deleted from the data set.

ARGTYP=
is used to describe the contents of the record identification field when adding a record to a VSAM data set (TYPOPER=NEWREC).

KEY
indicates that the record identification field contains a search key.

RBA
indicates that the record identification field contains a relative byte address.

Note: If the add request is part of a mass-insert operation, the ARGTYP parameter specified in the DFHFC TYPE=GETAREA macro instruction initiating the mass insert cannot be overridden.

NORESP, ERROR, DUPREC, INVREC, IOERROR, NOSPACE, NOTOPEN, and ILLOGIC (VSAM only)
are used to test the CICS/VS response to this request for file services. These operands can be specified in this macro instruction or in a DFHFC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for File Services" in this chapter.

RANDOMLY DELETE DATA FROM A DATA SET (VSAM ONLY)

The general format of a the DFHFC macro instruction to delete a record or group of records from a VSAM key-sequenced data set is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHFC │ TYPE=DELETE                                     │
│      │      │ [,DATASET=symbolic name]                         │
│      │      │ [,RIDADR=symbolic address]                       │
│      │      │ [,ARGTYP={KEY|RBA}]                              │
│      │      │ [,SRCHTYP={FKEQ|GKEQ}]                           │
│      │      │ [,NORESP=symbolic address]                       │
│      │      │ [,ERROR=symbolic address]                        │
│      │      │ [,DSIDER=symbolic address]                       │
│      │      │ [,NOTFND=symbolic address]                       │
│      │      │ [,INVREQ=symbolic address]                       │
│      │      │ [,IOERROR=symbolic address]                      │
│      │      │ [,NOTOPEN=symbolic address]                      │
│      │      │ [,ILLOGIC=symbolic address]                      │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=DELETE
indicates that deletion of a record or group of records from a VSAM key-sequenced data set is required.

DATASET=symbolic name
is the symbolic name of the data set to be accessed. The name must appear in the file control table (FCT). If this operand is omitted, the symbolic name is assumed to be in TCAFCDI.

RDIDADR=symbolic address
is the symbolic address of the user's record identification field that contains the key or relative byte address of the data to be deleted. If this operand is omitted, the symbolic address is assumed to be in TCAFCRI. (For further details, see "record identification Field" in Chapter 11.)

ARGTYP=
describes the contents of the record identification field of the data set.

KEY
indicates that the record identification field contains a search key.

RBA
indicates that the record identification field contains a relative byte address.

SRCHTYP=
specifies how the search key in the record identification field is to be used and is meaningful only when ARGTYP=KEY is specified or implied by default.

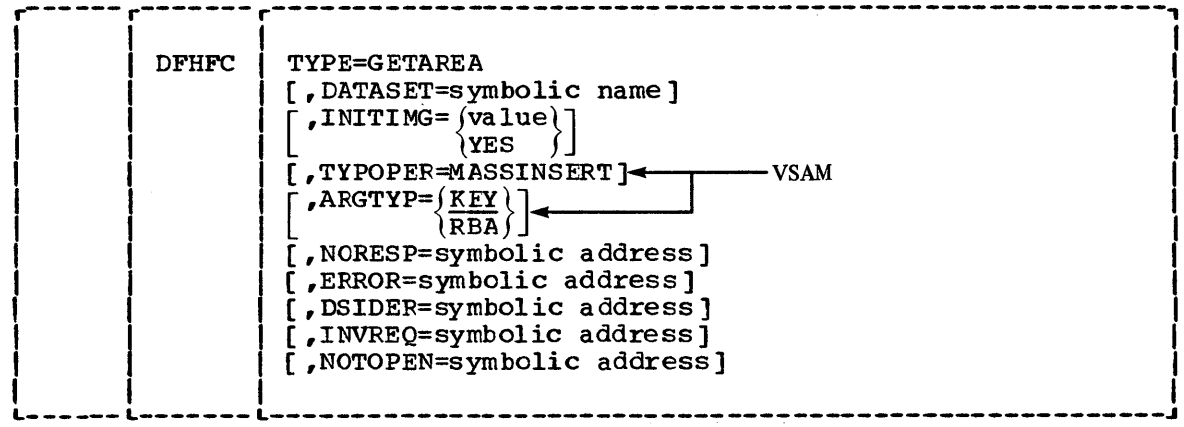
FKEQ
indicates that the search key is a full key and that only a record with an equal key satisfies the search.

GKEQ
indicates that the search key is a generic (partial) key, the binary length of which is specified in the first byte of the record identification field. All records whose keys begin with the search key are to be deleted. CICS/VS returns a count of the number of records deleted in TCAFCNRD.

NORESP, ERROR, DSIDER, NOTFND, INVREQ, IOERROR, NOTOPEN, and ILLOGIC
are used to test the CICS/VS response to this request for record deletion. These operands can be specified in this macro instruction or in a DFHFC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for File Services" in this chapter.

OBTAIN A FILE WORK AREA

The general format of the DFHFC macro instruction to obtain a file work area (FWA) is as follows:



where:

TYPE=GETAREA
indicates that CICS/VS is to acquire a FWA for the application program.

DATASET=symbolic name
is the symbolic name of the data set to be accessed. The name must appear in the file control table (FCT). If this operand is omitted, the symbolic name is assumed to be in TCAFCDI.

INITIMG=
specifies a one-byte (two-digit) hexadecimal initialization value for the FWA.

value
is a two-digit hexadecimal numeral to be used as the initialization value.

YES
indicates that the hexadecimal initialization value has been placed in TCASCIB.

If this operand is omitted, the FWA is initialized to EBCDIC blanks (X'40').

TYPOPER=MASSINSERT
is applicable to only VSAM data sets and specifies that the acquired FWA is to be used for a mass-insert operation. This ensures that the same FWA is used for subsequent DFHFC TYPE=PUT macro instructions adding new logical records with keys or relative byte addresses in ascending sequence to the data set. The FWA is made available to the application program after each DFHFC TYPE=PUT macro instruction. If storage initialization is specified in this DFHFC TYPE=GETAREA macro instruction, the FWA is reinitialized before each return to the application program. A mass-insert operation is terminated by a DFHFC TYPE=RELEASE macro instruction.

ARGTYP=
is applicable only when TYPOPER=MASSINSERT is specified and describes the contents of the record identification field(s)

that will be presented by the DFHFC TYPE=PUT macro instructions of this mass-insert operation.

KEY
indicates that the record identification field(s) contain a search key.

RBA
indicates that the record identification field(s) contain a relative byte address.

NORESP, ERROR, DSIDER, INVREQ, and NOTOPEN
are used to test the CICS/VS response to this request for file services. These operands can be specified in this macro instruction or in a DFHFC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for File Services" in this chapter.

RELEASE FILE STORAGE

The general format of the DFHFC macro instruction to release the storage areas acquired for file control operations is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHFC │ TYPE=RELEASE                                        │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,ERROR=symbolic address]                            │
│      │      │ [,INVREQ=symbolic address]                           │
│      │      │ [,IOERROR=symbolic address]                          │
│      │      │ [,ILLOGIC=symbolic address]◄──────VSAM                │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

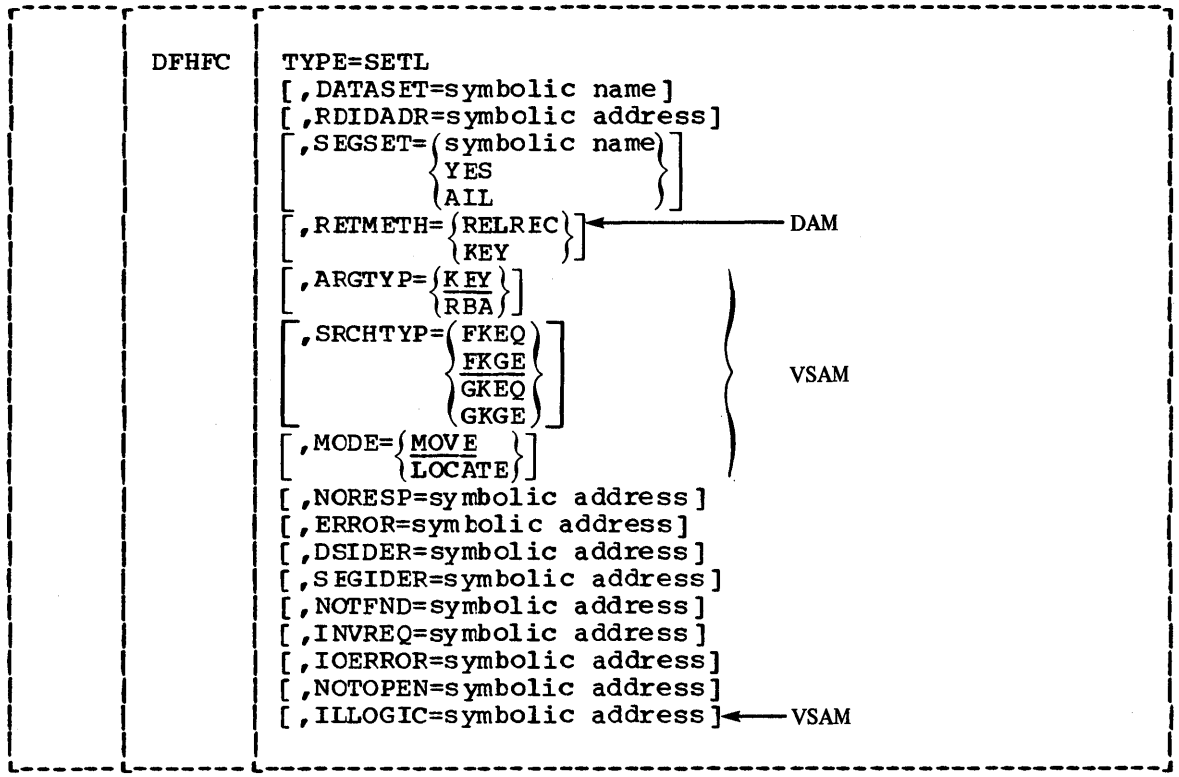
where:

TYPE=RELEASE
indicates that the FWA, FIOA, and/or VSWA acquired for file control operations and any exclusive control encumbrances are to be released.

NORESP, ERROR, INVREQ, IOERROR, and ILLOGIC (VSAM only)
are used to test the CICS/VS response to this request for release of exclusive control, data set encumbrances, and/or storage. These operands can be specified in this macro instruction or in a DFHFC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for File Services" in this chapter.

INITIATE SEQUENTIAL RETRIEVAL (BROWSING)

The general format of the DFHFC macro instruction to initiate a browse operation on a data set is as follows:



where:

TYPE=SETL
indicates that a browse operation is to be initiated.

DATASET=symbolic name
is the symbolic name of the data set to be accessed. The name must appear in the file control table (FCT). If this operand is omitted, the symbolic name is assumed to be in TCAFCDI.

RDIDADR=symbolic address
is the symbolic address of the user's record identification field that contains the specific or generic key as required by ISAM, the block reference as required by DAM, or the specific or generic key or relative byte address as required by VSAM. If this operand is omitted, the symbolic address is assumed to be in TCAFCRI. (For further details, see "record identification Field" in Chapter 11.)

SEGSET=
indicates that the data set specified in the DATASET operand contains segmented records and specifies the default segment set to be retrieved during a browse operation if no segment set name is specified in a subsequent DFHFC TYPE=GETNEXT macro instruction. This segment set name is used as the default segment set name throughout the browse operation unless altered by a DFHFC TYPE=RESETL macro instruction.

symbolic name
is the symbolic name of the default segment set. The name must have been defined in the associated segment control section of the FCT.

YES
indicates that the symbolic name of the default segment set has been placed in TCAFCSI.

ALL
indicates that the entire logical record in an unpacked and aligned format is desired.

If this operand is omitted, and the DFHFC TYPE=SETL macro instruction refers to a data set containing segmented records, the logical record is returned in its packed unaligned format.

DAM Data Set

RETMETH=
applies only to blocked DAM data sets and is used to specify the format of the logical record identification that is placed in the user's record identification field by CICS/VS each time the next logical record is retrieved in a browse operation.

RELREC
indicates that a one-byte binary relative record number is provided.

KEY
is valid only when records have embedded keys and indicates that the logical record key is to be provided.

VSAM Data Set

ARGTYP=
describes the contents of the record identification field of a VSAM data set.

KEY
is valid only when records have embedded keys and indicates that the logical record key is to be provided.

RBA
indicates that the record identification field contains a relative byte address.

SRCHTYP=
specifies how the search key in the record identification field is to be used in locating VSAM records. This operand is meaningful only when ARGTYP=KEY is specified or implied by default.

FKEQ
indicates that the search key is a full key and that only a record with an equal key satisfies the search.

FKGE
indicates that the search key is a full key and that the first record with a key equal to or greater than the search key satisfies the search.

GKEQ
indicates that the search key is a generic (partial) key, the binary length of which is specified in the first byte of the record identification field. A record whose key is equal to the search key (compared on only the number of bytes specified in the first byte of the record identification field) satisfies the search.

GKGE
indicates that the search key is a generic key and that the first record with a key equal to or greater than the search key (compared on only the number of bytes specified in the first byte of the record identification field) satisfies the search.

MODE=
is use to specify the processing mode for the browse operation.

MOVE
specifies move mode processing. Upon return to the application program, TCAFCAA contains the address of the FWA acquired for the browse operation. If the data set referred to contains variable-length records, the traditional LLØØ length field is included as part of each logical record retrieved during the browse.

LOCATE
specifies locate mode processing. Upon return to the application program, TCAFCAA contains the address of VSWA. The address of each record retrieved during the browse is placed at VSWAREA. If the data set referred to contains variable-length records, the traditional LLØØ length field is not retrieved as part of the logical record; instead, the length of the record is placed in VSWALEN. This parameter cannot be specified if the data set contains segmented records.

NORESP, ERROR, DSIDER, SEGIDER, NOTFND, INVREQ, IOERROR, NOTOPEN, and ILLOGIC (VSAM only)
are used to test the CICS/VS response to this request for file services. These operands can be specified in this macro instruction or in a DFHFC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for File Services" in this chapter.

RETRIEVE NEXT SEQUENTIAL RECORD

The general format of the DFHFC macro instruction to retrieve the next sequental record during a browse operation is shown below. This instruction can also be used to perform skip-sequential processing upon a VSAM data set (see "Retrieve Next Sequential Record (GETNEXT)" in Chapter 6).

```
DFHFC  TYPE=GETNEXT
       [,SEGSET={symbolic name}]
                {YES         }
                {ALL         }
       [,NORESP=symbolic address]
       [,ERROR=symbolic address]
       [,SEGIDER=symbolic address]
       [,NOTFND=symbolic address]
       [,INVREQ=symbolic address]
       [,IOERROR=symbolic address]
       [,NOTOPEN=symbolic address]
       [,ENDFILE=symbolic address]
       [,ILLOGIC=symbolic address]<------ VSAM
```

where:

TYPE=GETNEXT
: indicates that the next record in sequence is to be retrieved during a browse operation.

SEGSET=
: specifies the symbolic name of the segment set to be retrieved from the next record in sequence.

  symbolic name
  : is the symbolic name of the segment set to be retrieved. The name must have been defined in the associated segment control section of the FCT.

  YES
  : indicates that the symbolic name of the segment set has been placed in TCAFCSI.

  ALL
  : indicates that the entire logical record in an unpacked and aligned format is desired.

  If this operand is omitted and if the SEGSET operand was specified in the DFHFC TYPE=SETL macro instruction initiating the browse, the eight-character default segment set name, as specified in that instruction, is taken as the segment set name and returned at TCAFCSI upon normal completion of the GETNEXT.

  If this operand is omitted both here and in the DFHFC TYPE=SETL request and the browse operation is against a data set containing segmented records, the logical record is returned in its packed unaligned format.

  Note: If MODE=LOCATE was specified in the DFHFC TYPE=SETL macro instruction initiating the browse operation, inclusion of this operand is not permissible; an INVREQ (invalid request) condition occurs.

NORESP, ERROR, SEGIDER, NOTFND, INVREQ, IOERROR, NOTOPEN, ENDFILE, and ILLOGIC (VSAM only)
are used to test the CICS/VS response to this request for file services. These operands can also be specified in a DFHFC TYPE=CHECK macro instruction, and are discussed in detail under "Test Response to a Request for File Services" below.

TERMINATE SEQUENTIAL RETRIEVAL (BROWSING)

The general format of the DFHFC macro instruction to terminate a browse operation on a data set is as follows:

```
        DFHFC   TYPE=ESETL
                [,NORESP=symbolic address]
                [,ERROR=symbolic address]
                [,INVREQ=symbolic address]
                [,ILLOGIC=symbolic address]<------VSAM
```

where:

TYPE=ESETL
indicates that a browse operation is to be terminated.

NORESP, ERROR, INVREQ, and ILLOGIC (VSAM only)
are used to test the CICS/VS response to this request to terminate a browse operation. These operands can also be specified in a DFHFC TYPE=CHECK macro instruction and are discussed in detail under "Test Response to a Request for File Services" below.

RESET SEQUENTIAL RETRIEVAL

The general format of the DFHFC macro instruction to reset the search argument, default segment set name, and/or type of search argument (VSAM only) for a sequential retrieval (browse) operation is as follows:

```
        DFHFC   TYPE=RESETL
                [,SEGSET={symbolic name}]
                         {YES          }
                         {ALL          }
                [,ARGTYP={KEY}]                  )
                         {RBA}                   )
                [,SRCHTYP={FKEQ}]                ) VSAM
                          {FKGE}                 )
                          {GKEQ}                 )
                          {GKGE}                 )
                [,NORESP=symbolic address]
                [,ERROR=symbolic address]
                [,SEGIDER=symbolic address]
                [,NOTFND=symbolic address]
                [,INVREQ=symbolic address]
                [,IOERROR=symbolic address]
                [,NOTOPEN=symbolic address]
                [,ILLOGIC=symbolic address]<------VSAM
```

where:

TYPE=RESETL
indicates that the search argument, default segment set name, and/or type of search argument (VSAM only) established for a browse operation is to be changed.

SEGSET=
specifies a new default segment set name to be used in this browse operation.

symbolic name
is the symbolic name of the default segment set. The name must have been defined in the associated segment control section of the FCT.

YES
indicates that the symbolic name of the default segment set has been placed in TCAFCSI.

ALL
indicates that the entire logical record in an unpacked and aligned format is desired.

If this operand is omitted, the segment set name specified in a preceding SETL or RESETL macro instruction for this browse operation continues to be the segment set name.

VSAM Data Set

ARGTYP=
describes the contents of the record identification field of a VSAM data set.

KEY
indicates that the record identification field contains a search key.

RBA
indicates that the record identification field contains a relative byte address.

SRCHTYP=
specifies how the search key in the record identification field is to be used in locating VSAM records. This operand is meaningful only when ARGTYP=KEY is specified or implied by default.

FKEQ
indicates that the search key is a full key and that only a record with an equal key satisfies the search.

FKGE
indicates that the search key is a full key and that the first record with a key equal to or greater than the search key satisfies the search.

GKEQ
indicates that the search key is a generic (partial) key, the binary length of which is specified in the first byte of the record identification field. A record whose key is equal to the search key (compared on only the number of

bytes specified in the first byte of the record identification field) satisfies the search.

GKGE
: indicates that the search key is a generic key and that the first record with a key equal to or greater than the search key (compared on only the number of bytes specified in the first byte of the record identification field) satisfies the search.

NORESP, ERROR, SEGIDER, NOTFND, INVREQ, IOERROR, NOTOPEN, and ILLOGIC (VSAM only)
: are used to test the CICS/VS response to this request for file services. These operands can be specified in this macro instruction or in a DFHFC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for File Services" (below).

TEST RESPONSE TO A REQUEST FOR FILE SERVICES

The general format of the DFHFC macro instruction to test the CICS/VS response to a preceding DFHFC request for file services is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHFC │ TYPE=CHECK                                      │
│      │      │ [,NORESP=symbolic address]                       │
│      │      │ [,ERROR=symbolic address]                        │
│      │      │ [,DSIDER=symbolic address]                       │
│      │      │ [,SEGIDER=symbolic address]                      │
│      │      │ [,NOTFND=symbolic address]                       │
│      │      │ [,DUPREC=symbolic address]                       │
│      │      │ [,INVREQ=symbolic address]                       │
│      │      │ [,IOERROR=symbolic address]                      │
│      │      │ [,DUPDS=symbolic address]                        │
│      │      │ [,NOSPACE=symbolic address]                      │
│      │      │ [,NOTOPEN=symbolic address]                      │
│      │      │ [,ENDFILE=symbolic address]                      │
│      │      │ [,ILLOGIC=symbolic address]◄──────VSAM           │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
: indicates that the CICS/VS response to a request for file services is to be checked.

NORESP=symbolic address
: specifies the entry label of the user-written routine to which control is to be passed if no error occurs on a file operation. NORESP signifies "normal response."

  The field TCAFCAA in the TCA of the task contains:

  - The address of an FIOA after a read-only unsegmented GET against an unblocked data set or a blocked DAM data set if deblocking is not requested

- The address of an FWA after a GET against a blocked data set, a GET segmented, a GET for update, a GETAREA, SETL, GETNEXT, or RESETL
- The address of a VSWA after a locate-mode GET or SETL against a VSAM data set and after a GETNEXT or RESETL for a browse operation initiated by a locate-mode SETL
- Meaningless information after a PUT, DELETE, RELEASE, or ESETL

ERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if any error occurs on a file operation. The CICS/VS response code should be further interrogated in this user-written routine. The contents of the field TCAFCAA are as described below for specific error conditions.

DSIDER=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the data set specified by the DATASET operand (at TCAFCDI) cannot be located in the FCT. The error also occurs if the data set specified in the INDEX operand of a DFHFC TYPE=GET macro instruction (available at TCAFCAI), or any of the lower-level data sets in the indirect accessing hierarchy, cannot be found in the FCT. DSIDER signifies "data set identification error." The contents of TCAFCAA are not meaningful.

SEGIDER=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the segment set specified by the SEGSET operand (at TCAFCSI) cannot be located for this data set in the FCT. SEGIDER signifies "segment set identification error."

The field TCAFCAA contains:

- Zero for GET, SETL, or RESETL
- The address of the FWA for GETNEXT

For RESETL, the browse operation will have been terminated.

NOTFND=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an attempt to retrieve or delete a record based on the search argument provided is unsuccessful. NOTFND signifies "record not found."

TCAFCAA contains:

- The address of an FIOA if the request was a GET against an ISAM or a DAM data set
- The address of a VSWA if the request was a GET, DELETE, SETL, RESETL, or GETNEXT request using skip-sequential against a VSAM data set

The application programmer should be aware of the following considerations:

- Except for RESETL or GETNEXT, the area returned should be released using DFHFC TYPE=RELEASE when any interrogation of the area is complete.

- For SETL, the browse operation was not initiated.

- For RESETL or GETNEXT, the browse operation is still active, but the position of the data set has been destroyed. A RESETL should be issued to reestablish the position in the data set. If move mode was specified or implied in the SETL request initiating the browse operation, the FWA representing the browse must be used for the RESETL; if locate mode was specified in the SETL request, the VSWA must be used.

DUPREC=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an attempt is made to add a record to a data set in which a record with the same key already exists. DUPREC signifies "duplicate record."

TCAFCAA contains:

- The address of an FIOA if the PUT request is against an ISAM or DAM data set

- The address of a VSWA if the PUT request is against a VSAM data set

The FWA will have been released. After any interrogation of the area returned is complete, the area should be released using DFHFC TYPE=RELEASE.

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the attempted file operation is not provided for or allowed according to the data set entry specification in the FCT. INVREQ signifies "invalid request."

TCAFCAA contains:

- The address of the appropriate FCT data set entry if the request is not allowed according to that FCT entry

- Zero if the request is truly invalid or if the code to support the request has not been generated into the CICS/VS file control program

IOERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an unusual event occurs during a file operation. When an I/O event error code is not covered by one of the CICS/VS error classes (for example, by NOSPACE or NOTFND), it is considered to be an I/O error.

TCAFCAA contains:

- The address of an FIOA if the request is against an ISAM or a DAM data set

- The address of a VSWA if the request is against a VSAM data set

The application programmer should be aware of the following conditions:

- For an ISAM or DAM data set, the actual error codes may be checked in the FIOA by the user's routine (FCFIOEX for ISAM and FCFIOBEX for DAM). Because of access method

and operating system dependencies, checks for these codes may have a limiting effect on the usability of an application program in varying environments, particularly if migration from CICS/DOS/VS to CICS/OS/VS becomes desirable.

- For a VSAM data set, the actual error codes may be checked in the request parameter list (RPL) located at VSWARPL. The error code is at VSWAERRC, and the return code is at VSWARTNC. Because of access method and operating system dependencies, checks for these codes may have a limiting effect on the usability of an application program in varying environments, particularly if migration from CICS/DOS/VS to CICS/OS/VS becomes desirable.

- For RESETL or GETNEXT the browse operation is still active. A RESETL using the record identification for the next record required should be issued to reestablish the position in the data set. If move mode was specified or implied in the initiating SETL request, the FWA representing the browse operation must be used for the RESETL; if locate mode was specified in the SETL request, the VSWA must be used.

- For PUT, the FWA will have been released.

After any interrogation of the area returned is complete, the area should be released using DFHFC TYPE=RELEASE.

DUPDS=symbolic address
specifies the entry label of the user-written routine to which control is to be transferred if the record retrieved on an indirect access is from a duplicates data set rather than from the primary data set. The user's routine can include provisions for processing the duplicate record. DUPDS signifies "duplicates data set."

TCAFCAA contains:

- The address of an FIOA if the duplicates data set is unblocked

- The address of an FWA if the duplicates data set is blocked

NOSPACE=symbolic address
specifies the entry label of the user-written routine to which control is to be transferred if no direct access space is available for adding records to a data set. This error condition is not applicable when adding records to a fixed-length DAM data set that does not contain keys. TCAFCAA contains the address of an FWA containing the record to be added. This FWA may be at a different storage location than the FWA passed with the PUT request.

NOTOPEN=symbolic address
specifies the entry label of the user-written routine to which control is to be transferred if the requested data set is not open. This error condition can occur in response to any file service request except RELEASE and ESETL, because a data base data set can be closed dynamically at any time without regard to outstanding activity on the data set. The contents of TCAFCAA are not meaningful.

ENDFILE=symbolic address
specifies the entry label of the user-written routine to which control is to be transferred if an end-of-file condition is detected during the sequential retrieval (browse) of records in a data set. This condition can occur only in response to a GETNEXT request. TCAFCAA contains the address of the FWA for the browse operation if move mode was specified or implied in the SETL request. TCAFCAA contains the address of the VSWA that represents the browse if locate mode was specified.

VSAM Data Set

ILLOGIC=symbolic address
specifies the entry label of the user-written routine to which control is to be transferred if a VSAM error that does not fall within one of the other CICS/VS response categories occurs. TCAFCAA contains the address of a VSWA. The user's routine may check the actual logical error codes in the RPL which is at VSWARPL. The error code is at VSWAERRC, and the return code is at VSWARTNC.

After an interrogation of the area returned, this area should be freed by means of a DFHFC TYPE=RELEASE macro instruction.

DFHTD MACRO INSTRUCTION

The transient data macro instruction (DFHTD) is used to request transient data services for an intrapartition or extrapartition data set as explained below.

DISPOSE OF DATA

The general format of the DFHTD macro instruction to direct transient data to a predefined symbolic destination is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHTD │  TYPE=PUT                                           │
│      │      │ [,DESTID=symbolic name]                              │
│      │      │ [,TDADDR=symbolic address]                           │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,IDERROR=symbolic address]                          │
│      │      │ [,IOERROR=symbolic address]                          │
│      │      │ [,NOTOPEN=symbolic address]                          │
│      │      │ [,NOSPACE=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=PUT
indicates that data is to be written to an intrapartition or extrapartition data set.

DESTID=symbolic name
specifies the symbolic name of the destination to which the data is to be routed and queued. This name must appear in the destination control table (DCT). If this operand is omitted, the symbolic name of the destination is assumed to be in TCATDDI.

TDADDR=symbolic name
specifies the symbolic name of the output area containing data to be written (for intrapartition data and variable-length extrapartition data, the first four bytes of this area must contain the length of the record). If this operand is omitted, the address of the output area is assumed to be in TCATDAA.

NORESP, IDERROR, IOERROR, NOTOPEN, and NOSPACE
are used to test the CICS/VS response to this request for transient data output. These operands can be specified in this macro instruction or in a DFHTD TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Transient Data Services" in this chapter.

ACQUIRE QUEUED DATA

The general format of the DFHTD macro instruction to acquire queued data on an extrapartition or intrapartition destination is shown below. The address of the data acquired is returned at TCATDAA.

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHTD │  TYPE=GET                                            │
│      │       │ [,DESTID=symbolic address]                           │
│      │       │ [,QUEBUSY=symbolic address]◄──────CICS/OS/VS only     │
│      │       │ [,NORESP=symbolic address]                           │
│      │       │ [,QUEZERO=symbolic address]                          │
│      │       │ [,IDERROR=symbolic address]                          │
│      │       │ [,IOERROR=symbolic address]                          │
│      │       │ [,NOTOPEN=symbolic address]                          │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=GET
indicates that data is to be read from an intrapartition or extrapartition data set.

DESTID=symbolic name
specifies the symbolic name of the destination from which queued data is to be read. The name must appear in the destination control table (DCT). If this operand is omitted, the symbolic name is assumed to be in TCATDDI.

QUEBUSY=symbolic address
specifies the symbolic address of the routine to receive control if the input request attempts to access a record on an input intrapartition queue that has been enqueued upon for output. If this operand is not specified, the task issuing the request waits until the queue is no longer being used for output.

NORESP, QUEZERO, IDERROR, IOERROR, and NOTOPEN
are used to test the CICS/VS response to this request for queued data. These operands can be specified in this macro instruction or in a DFHTD TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Transient Data Services" in this chapter.

CONTROL PROCESSING OF EXTRAPARTITION DATA SET (MAGNETIC TAPE ONLY)

The general format of the DFHTD macro instruction to create a "forced end of volume" situation on an extrapartition magnetic tape data set is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHTD │  TYPE=FEOV                                          │
│      │      │ [,DESTID=symbolic name]                              │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,IDERROR=symbolic address]                          │
│      │      │ [,NOTOPEN=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=FEOV
: indicates that a magnetic tape reel is to be rewound and unloaded; output labels are to be created as required and new input labels verified according to host operating system forced-end-of-volume processing. CICS/VS operation is halted, and the next tape reel must be loaded before CICS/VS operation is resumed.

DESTID=symbolic name
: specifies the symbolic name of the destination against which "forced end of volume" is to be applied. The name must appear in the destination control table (DCT). If this operand is omitted, the symbolic name is assumed to be in TCATDDI.

NORESP, IDERROR, and NOTOPEN
: are used to test the CICS/VS response to this request for transient data services. These operands can be specified in this macro instruction or in a DFHTD TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Transient Data Services" in this chapter.

PURGE INTRAPARTITION DATA

The general format of the DFHTD macro instruction to purge all data associated with a particular intrapartition destination (queue) is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHTD │  TYPE=PURGE                                         │
│      │      │ [,DESTID=symbolic name]                              │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,IDERROR=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=PURGE
: indicates that all storage associated with a particular intrapartition destination is to be deallocated.

DESTID=symbolic name
: specifies the symbolic name of the destination associated with the data to be purged. The name must appear in the destination

control table (DCT). If this operand is omitted, the symbolic name is assumed to be in TCATDDI.

NORESP and IDERROR
are used to test the CICS/VS response to this request for transient data services. These operands can be specified in this macro instruction or in a DFHTD TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Transient Data Services" in this chapter.

TEST RESPONSE TO A REQUEST FOR TRANSIENT DATA SERVICES

The general format of the DFHTD macro instruction to test the CICS/VS response to a request for transient data services is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHTD │  TYPE=CHECK                                     │
│      │      │ [,NORESP=symbolic address]                       │
│      │      │ [,QUEZERO=symbolic address]                      │
│      │      │ [,IDERROR=symbolic address]                      │
│      │      │ [,IOERROR=symbolic address]                      │
│      │      │ [,NOTOPEN=symbolic address]                      │
│      │      │ [,NOSPACE=symbolic address]                      │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
indicates that the CICS/VS response to a preceding DFHTD macro instruction is to be checked.

NORESP=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if no error occurs during a data set (file) operation. NORESP signifies "normal response."

QUEZERO=symbolic address
specifies the entry label of the user-written routine to which control is to be passed when the destination (queue) accessed by a DFHTD TYPE=GET macro instruction is empty.

IDERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the symbolic destination referred to by a DFHTD macro instruction cannot be found.

IOERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an input/output error occurs on a data record and the data record in error is skipped. Transient data returns an IOERROR indication as long as the queue can be read; a QUEZERO response is returned when the queue cannot be read, in which case, the user may attempt a restart.

NOTOPEN=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if a destination is closed.

NOSPACE=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if no more space exists on a particular

intrapartition queue or if the write request cannot be serviced. If the NOSPACE response is received, no more data should be written to the queue, because it may be lost.

DFHTS MACRO INSTRUCTION

The temporary storage macro instruction (DFHTS) is used to request temporary storage services as explained below.

STORE TEMPORARY DATA AS A SINGLE UNIT OF INFORMATION

The general format of the DFHTS macro instruction to store a single unit of information as temporary data on main or auxiliary storage (that is, as though using a "scratch pad") is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHTS │ TYPE=PUT                                             │
│      │       │ [,TYPOPER=REPLACE]                                   │
│      │       │ [,DATAID=name]                                       │
│      │       │ [,TSDADDR={symbolic address}]                        │
│      │       │           {YES            }                          │
│      │       │ [,STORFAC={AUXILIARY}]                               │
│      │       │           {MAIN     }                                │
│      │       │ [,COND=YES]                                          │
│      │       │ [,NOSPACE=symbolic address                           │
│      │       │ [,NORESP=symbolic address]                           │
│      │       │ [,IOERROR=symbolic address]                          │
│      │       │ [,INVREQ=symbolic address]                           │
│      │       │ [,ERROR=symbolic address]                            │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=PUT
: indicates that a single unit of information is to be placed in temporary storage.

TYPOPER=REPLACE
: indicates that the current data is to be released and replaced with the data provided. If current data does not exist (DATAID cannot be found), the data provided is placed in temporary storage as in a normal PUT without TYPOPER=REPLACE specified.

DATAID=name
: specifies the unique alphameric name, up to eight characters in length, to be assigned to the temporary data to be stored. If this operand is omitted, the name is assumed to be in TCATSDI.

  Note: The application program should not construct a DATAID beginning with any of the hexadecimal characters FA through FF. Use of these characters for this purpose is reserved for CICS/VS.

TSDADDR=
: specifies the symbolic address of the data portion (including the LLØØ field) of the area in which the temporary data is stored.

  symbolic address
  : is the symbolic address of the data portion of the storage area that contains the temporary data.

YES
indicates that the symbolic address of the data portion of the storage area has been placed in TCATSDA by the application programmer.

If this operand is omitted, the appropriate symbolic address is assumed to be in TCATSDA.

STORFAC=
specifies the type of storage to be used for the temporary data.

AUXILIARY
indicates that the data is to be placed in auxiliary storage on a direct access storage device.

MAIN
indicates that the data is to be placed in main storage.

If this operand is omitted, AUXILIARY is assumed.

COND=YES
indicates that control is to be returned to the application program when the request cannot be satisfied immediately because sufficient space is not available on the temporary storage data set. If this operand is omitted, the requesting task is suspended when no space is available and is resumed when the space becomes available. Space becomes available as it is released by other tasks in the system.

NOSPACE, NORESP, IOERROR, INVREQ, and ERROR
are used to test the CICS/VS response to this request for storage of temporary data. These operands can be specified in this macro instruction or in a DFHTS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Temporary Storage Services" in this chapter.

STORE DATA TO A TEMPORARY STORAGE MESSAGE SET

The general format of the DFHTS macro instruction to cause an entry to be written to a temporary storage message set is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHTS │ TYPE=PUTQ                                            │
│      │       │ [,TYPOPER=REPLACE]                                   │
│      │       │ [,DATAID=name]                                       │
│      │       │ [,TSDADDR={symbolic address}]                        │
│      │       │           {YES             }                         │
│      │       │ [,STORFAC={AUXILIARY}]                               │
│      │       │           {MAIN     }                                │
│      │       │ [,ENTRY={n  }]                                       │
│      │       │         {yes}                                        │
│      │       │ [,COND=YES]                                          │
│      │       │ [,NOSPACE=symbolic address]                          │
│      │       │ [,NORESP=symbolic address]                           │
│      │       │ [,IOERROR=symbolic address]                          │
│      │       │ [,INVREQ=symbolic address]                           │
│      │       │ [,ENERROR=symbolic address]                          │
│      │       │ [,ERROR=symbolic address]                            │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=PUTQ
indicates that a logical record is to be written to a temporary storage message set.

TYPOPER=REPLACE
indicates that the specified ENTRY within the message set is to be released and replaced with the data provided. Whenever REPLACE is specified, the ENTRY operand must also be coded. If the message set does not exist (DATAID cannot be found), the data provided is placed in temporary storage as in a normal PUTQ without TYPOPER=REPLACE specified.

DATAID=name
specifies the unique alphameric name, up to eight characters in length, of the temporary storage message set to which this record is to be written. If this operand is omitted, the name is assumed to be in TCATSDI.

Note: The application program should not construct a DATAID beginning with any of the hexadecimal characters FA through FF. Use of these characters for this purpose is reserved for CICS/VS.

TSDADDR=
specifies the symbolic address of the data portion (including the LLØØ field) of the area in which the temporary data is stored.

symbolic address
is the symbolic address of the data portion of the storage area that contains the temporary data.

YES
indicates that the symbolic address of the data portion of the storage area has been placed in TCATSDA by the application programmer.

If this operand is omitted, the appropriate symbolic address is assumed to be in TCATSDA.

STORFAC=
specifies the type of storage used for the temporary storage message set.

AUXILIARY
indicates that the temporary storage message set is on auxiliary storage on a direct access storage device.

MAIN
indicates that the temporary storage message set is in main storage.

If this operand is omitted, AUXILIARY is assumed.

ENTRY=
specifies the number, relative to one, of the logical record to be updated from the message set.

n
is a decimal numeral to be taken as the relative number of the logical record to be updated. This number may be the number of any entry that has been written to the temporary storage message set.

YES
indicates that the number of the logical record to be updated is in TCATSRN, a two-byte binary field.

The ENTRY operand must always be specified whenever TYPOPER=REPLACE is specified and vice versa.

COND=YES
indicates that control is to be returned to the application program when the request cannot be satisfied immediately because sufficient space is not available on the temporary storage data set. If this operand is omitted, the requesting task is suspended when no space is available and is resumed when the space becomes available. Space becomes available as it is released by other tasks in the system.

NOSPACE, NORESP, IOERROR, INVREQ, ENERROR, and ERRCR
are used to test the CICS/VS response to this request for storage of temporary data. These operands can be specified in this macro instruction or in a DFHTS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Temporary Storage Services" in this chapter.

RETRIEVE A SINGLE UNIT OF TEMPORARY DATA

The general format of the DFHTS macro instruction to retrieve a single unit of temporary data is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHTS │ TYPE=GET                                             │
│      │       │ [,DATAID=name]                                       │
│      │       │ [,TSDADDR={symbolic address}]                        │
│      │       │           {YES             }                       │
│      │       │ [,RELEASE={YES}]                                     │
│      │       │           {NO }                                      │
│      │       │ [,NORESP=symbolic address]                           │
│      │       │ [,IDERROR=symbolic address]                          │
│      │       │ [,IOERROR=symbolic address]                          │
│      │       │ [,INVREQ=symbolic address]                           │
│      │       │ [,ERROR=symbolic address ]                           │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=GET
indicates that a single unit of information is to be retrieved from temporary storage.

DATAID=name
specifies the name assigned to the temporary data at the time it was placed in temporary storage. If this operand is omitted, the name is assumed to be in TCATSDI.

TSDADDR=
indicates that the application programmer has selected a storage area in which the temporary data is to be placed and identifies it.

symbolic address
is the symbolic address of the application-programmer-selected storage area in which the data is to be placed.

YES
indicates that the symbolic address of the storage area to be used for the retrieved data has been placed in TCATSDA by the application programmer.

If this operand is omitted, temporary storage control obtains a storage area, moves temporary data into the area, and returns the address of the data moved into the area (beginning with the LLØØ field) in TCATSDA to the application program. To determine the address of the storage area, a value of 8 should be subtracted from the address in TCATSDA.

RELEASE=
specifies the disposition of the temporary data following the move operation.

YES
the data and storage area used for the data are to be released after this operation.

NO
the data is to be retained, available for subsequent similar reference.

NORESP, IDERROR, IOERROR, INVREQ, and ERROR
are used to test the CICS/VS response to this request for acquisition of temporary data. These operands can be specified in this macro instruction or in a DFHTS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Temporary Storage Services" in this chapter.

RETRIEVE DATA FROM A TEMPORARY STORAGE MESSAGE SET

The general format of the DFHTS macro instruction to retrieve a logical record from a temporary storage message set is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHTS │ TYPE=GETQ                                       │
│      │      │ [,DATAID=name]                                   │
│      │      │ [,TSDADDR={symbolic address}]                    │
│      │      │           {YES            }                      │
│      │      │ [,ENTRY={n  }]                                   │
│      │      │         {YES}                                    │
│      │      │ [,NORESP=symbolic address]                       │
│      │      │ [,IDERROR=symbolic address]                      │
│      │      │ [,IOERROR=symbolic address]                      │
│      │      │ [,INVREQ=symbolic address]                       │
│      │      │ [,ENERROR=symbolic address]                      │
│      │      │ [,ERROR=symbolic address]                        │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=GETQ
indicates that a logical record is to be retrieved from a temporary storage message set.

DATAID=name
specifies the name assigned to the temporary storage message set from which the record is to be retrieved. If this operand is omitted, the name is assumed to be in TCATSDI.

TSDADDR=
indicates that the application programmer has selected a storage area for the logical record to be retrieved and identifies it.

symbolic address
is the symbolic address of the application programmer selected storage area in which the record is to be placed.

YES
indicates that the symbolic address of the storage area into which the logical record is to be moved has been placed in TCATSDA by the application programmer.

If this operand is omitted, temporary storage control is responsible for selection of the storage area for the logical record to be retrieved. One logical record is moved from the temporary storage message set to this area. The address of the data moved to the area (beginning with the LLØØ field) is returned in TCATSDA to the application program. To determine the address of the storage area, a value of 8 should be subtracted from the address in TCATSDA.

ENTRY=
specifies the number, relative to one, of the logical record to be retrieved from the message set.

n
is a decimal numeral to be taken as the relative number of the logical record to be retrieved. This number may be the number of any entry that has been written to the temporary storage message set.

YES
indicates that the number (in binary) of the logical record to be retrieved is in TCATSRN, a two-byte field.

If this operand is omitted, CICS/VS retrieves (1) the first logical record from the message set, for the first retrieval request, or (2) the next sequential logical record following the last-retrieved record, for other than the first request. In the latter case, the relative record number is returned in TCATSRN.

NORESP, IDERROR, IOERROR, INVREQ, ENERROR, and ERROR
are used to test the CICS/VS response to this request for temporary data. These operands can be specified in this macro instruction or in a DFHTS TYPE=CHECK macro instruction. The meaning of each operand is discussed under "Test Response to a Request for Temporary Storage Services" in this chapter.

RELEASE A SINGLE UNIT OF TEMPORARY DATA

The general format of the DFHTS macro instruction to release a single unit of data placed in temporary storage by means of a DFHTS TYPE=PUT macro instruction is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────┐
│      │       │                                                  │
│      │ DFHTS │ TYPE=RELEASE                                     │
│      │       │ [,DATAID=name]                                   │
│      │       │ [,NORESP=symbolic address]                       │
│      │       │ [,IDERROR=symbolic address]                      │
│      │       │ [,INVREQ=symbolic address]                       │
│      │       │ [,ERROR=symbolic address]                        │
│      │       │                                                  │
└──────┴───────┴──────────────────────────────────────────────────┘
```

where:

TYPE=RELEASE
indicates that a single unit of temporary data is to be released.

DATAID=name
specifies the name assigned to the temporary data at the time it was placed in temporary storage. If this operand is omitted, the name is assumed to be in TCATSDI.

NORESP, IDERROR, INVREQ, and ERROR
are used to test the CICS/VS response to this request for temporary storage services. These operands can be specified in this macro instruction or in a DFHTS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Temporary Storage Services" in this chapter.

PURGE A TEMPORARY STORAGE MESSAGE SET

The general format of the DFHTS macro instruction to release data saved as a temporary storage message set (that is, in response to DFHTS TYPE=PUTQ macro instructions) is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────┐
│      │       │                                                  │
│      │ DFHTS │ TYPE=PURGE                                       │
│      │       │ [,DATAID=name]                                   │
│      │       │ [,NORESP=symbolic address]                       │
│      │       │ [,IDERROR=symbolic address]                      │
│      │       │ [,INVREQ=symbolic address]                       │
│      │       │ [,ERROR=symbolic address]                        │
│      │       │                                                  │
└──────┴───────┴──────────────────────────────────────────────────┘
```

where:

TYPE=PURGE
indicates that a temporary storage message set is to be released.

DATAID=name
specifies the name assigned to the temporary storage message set and used when logical records were added to the set. If this operand is omitted, the name is assumed to be in TCATSDI.

NORESP, IDERROR, INVREQ, and ERRCR
are used to test the CICS/VS response to this request for temporary storage services. These operands can be specified in this macro instruction or in a DFHTS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Temporary Storage Services" (below).

TEST RESPONSE TO A REQUEST FCR TEMPORARY STORAGE SERVICES

The general format of the DFHTS macro instruction to test the CICS/VS response to a request for temporary storage services is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────┐
│      │       │                                                  │
│      │ DFHTS │ TYPE=CHECK                                       │
│      │       │ [,NOSPACE=symbolic address]                      │
│      │       │ [,NORESP=symbolic address]                       │
│      │       │ [,IDERROR=symbolic address]                      │
│      │       │ [,IOERROR=symbolic address]                      │
│      │       │ [,INVREQ=symbolic address]                       │
│      │       │ [,ENERROR=symbolic address]                      │
│      │       │ [,ERROR=symbolic address]                        │
│      │       │                                                  │
└──────┴───────┴──────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
indicates that the CICS/VS response to a preceding DFHTS macro instruction is to be checked.

NOSPACE=symbolic address
specifies the entry label of the user-written routine to which control is to be passed when insufficient space is available on the temporary storage data set to contain the data in a PUT or PUTQ request. The user-written NOSPACE routine is passed control only if COND=YES is also specified in the PUT or PUTQ request.

NORESP=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if no errors occur during temporary storage processing. NORESP signifies "normal response."

IDERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the symbolic destination identification referred to by a GET, GETQ, RELEASE, or PURGE request cannot be found in either main storage or auxiliary storage.

IOERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an unrecoverable input/output error occurs.

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if (1) a PUT or PUTQ request refers to data whose length is equal to zero or greater than the control interval size of the auxiliary data set minus 32, or (2) the request is otherwise determined to be invalid.

ENERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the entry number specified or implied is invalid (that is, not within the limits of the message set).

ERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an error occurs and the corresponding specific error routine operand (for example, IDERROR) has not been specified.

DFHSC MACRO INSTRUCTION

The storage control macro instruction (DFHSC) is used to obtain and initialize main storage, or to release main storage, as explained below.

OBTAIN AND INITIALIZE MAIN STORAGE

The general format of the DFHSC macro instruction to obtain main storage and initialize the area obtained, if desired, is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHSC │ TYPE=GETMAIN                                        │
│      │      │ [,INITIMG={number}]                                  │
│      │      │           {YES   }                                   │
│      │      │ [,NUMBYTE=number]                                    │
│      │      │ [,COND={YES                    }]                    │
│      │      │        {(YES,symbolic address)}                      │
│      │      │        {(NO,symbolic address) }                      │
│      │      │ ,CLASS={TERMINAL or TERM}                            │
│      │      │        {USER             }                           │
│      │      │        {TRANSDATA or TD  }                           │
│      │      │        {TEMPSTRG or TS   }                           │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=GETMAIN
indicates that a main storage area is to be acquired.

INITIMG=
is an optional operand that can be used to initialize the acquired storage area to the bit configuration desired.

number
is a two-digit hexadecimal numeral indicating the bit configuration desired.

YES
indicates that the desired bit configuration is in TCASCIB.

NUMBYTE=number
is a decimal numeral up to 65520 (65515 when CLASS=TERMINAL) indicating the size, in bytes, of the storage area being requested; if omitted, the number of bytes is assumed to be stored in binary form in TCASCNB. A zero data length is not allowed for a DFHSC TYPE=GETMAIN macro instruction.

Note: Depending upon the class of storage specified (see the CLASS operand below), CICS/VS storage management automatically increments the amount of storage requested to allow for the

storage accounting field and other control information. For CLASS=USER and CLASS=TERMINAL (TIOA) storage, the exact number of bytes required should be specified. For CLASS=TRANSDATA (TDIA and TDOA) and CLASS=TEMPSTRG (TSIOA) storage, the amount requested must include four additional bytes to allow for a portion of CICS/VS control information, namely, the length (LL00) field at the beginning of the area.

COND=

is an optional operand that ensures that control is returned to the application program, whether or not the requested storage area is acquired.

YES
indicates that control is to be given to the instruction immediately following the macro expansion for the DFHSC TYPE=GETMAIN macro instruction in the application program. To determine whether the requested storage area was acquired, the application program must examine TCASCSA, which is set to binary zeros if the request cannot be satisfied.

(YES,symbolic address)
causes a branch to the location specified by the symbolic address if the requested storage was acquired; otherwise, control is returned to the instruction immediately following the macro expansion for the DFHSC TYPE=GETMAIN macro instruction in the application program.

(NO,symbolic address)
causes a branch to the location specified by the symbolic address if the requested storage was not acquired; otherwise, control is returned to the instruction immediately following the macro expansion for this macro instruction in the application program.

CLASS=

specifies the class of the storage to be acquired.

TERMINAL or TERM
indicates that the storage area is to be used as a terminal input/output area (TIOA), which is chained to the terminal control table terminal entry (TCTTE). All requests for storage related to terminal input/output must specify this class.

USER
indicates that the storage area is to be associated with the application program and used by that program. This area is chained to the TCA associated with the requesting task.

TRANSDATA or TD
indicates that the storage area is to be used for transient data record storage (a TDIA or TDOA). This area is chained to the TCA associated with the requesting task and is used by transient data control.

TEMPSTRG or TS
indicates that the storage area is to be used as a temporary storage input/output area (TSIOA). This area is chained to the TCA associated with the requesting task and is used by temporary storage control.

Note: USER, TRANSDATA, and TEMPSTRG specifications have essentially the same effect. The advantage of using CLASS=TRANSDATA or CLASS=TEMPSTRG when either is appropriate is

that the specification serves as documentation both in the program and in the class code of the storage accounting field for the area.

RELEASE MAIN STORAGE

The general format of the DFHSC macro instruction to release main storage is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHSC │ TYPE=FREEMAIN                                       │
│      │      │  [,RELEASE=ALL]                                      │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=FREEMAIN
indicates that previously acquired main storage is to be released.

RELEASE=ALL
indicates that all main storage acquired by means of DFHSC TYPE=GETMAIN,CLASS=TERMINAL macro instructions is to be released.

The use of the RELEASE=ALL operand is restricted during basic mapping support (BMS) output operations that have an OUT disposition; this restriction preserves the terminal storage used by BMS. Once a DFHBMS macro instruction with an OUT disposition has been issued, the application program must not issue a DFHSC TYPE=FREEMAIN,RELEASE=ALL macro instruction until either a DFHBMS TYPE=PAGEOUT or DFHBMS TYPE=PURGE macro instruction has been issued.

If this operand is not specified, only one storage area can be released by a DFHSC TYPE=FREEMAIN macro instruction; the address of that area must be in TCASCSA and must be the main storage address returned as a result of a previously issued DFHSC TYPE=GETMAIN macro instruction.

DFHPC MACRO INSTRUCTION

The program control macro instruction (DFHPC) provides for program communication within CICS/VS as explained below.

PASS PROGRAM CONTROL ANTICIPATING RETURN

The general format of the DFHPC macro instruction to pass program control to an application program at the next lower logical level is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHPC │   TYPE=LINK                                         │
│      │      │  [,PROGRAM=name]                                     │
│      │      │  [,COND=YES]                                         │
│      │      │  [,NORESP=symbolic address]                          │
│      │      │  [,PGMIDER=symbolic address]                         │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=LINK
indicates that program control is to be passed to an application program at the next lower logical level and subsequently returned to the application program issuing this macro instruction.

PROGRAM=name
is the identification of the program to which control is to be passed; if omitted, the identification is assumed to be in TCAPCPI. TCAPCPI is an eight-character field; identifications less than eight characters must be padded right with blanks.

COND=YES
indicates that control is to be returned to the program issuing this macro instruction if the requested program cannot be found in the PPT. If this operand is omitted and the requested program cannot be found or is disabled, the transaction is abnormally terminated with an APCT ABEND code.

NORESP and PGMIDER
are used to test the CICS/VS response to this request for program management services. These operands can be specified in this macro instruction or in a DFHPC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Program Services."

TRANSFER PROGRAM CONTROL

The general format of the DFHPC macro instruction to pass (transfer) program control to an application program at the same logical level is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHPC │   TYPE=XCTL                                     │
│      │      │  [,PROGRAM=name]                                 │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=XCTL
indicates that program control is to be transferred to an application program at the same logical level and that no subsequent return to the program requesting the transfer is required; the program from which this request is issued is released.

PROGRAM=name
is the identification of the program to which control is to be passed; if omitted, the identification is assumed to be in TCAPCPI. TCAPCPI is an eight-character field; identifications less than eight characters must be padded right with blanks. If the requested program cannot be found or is disabled, the task is abnormally terminated with an APCT ABEND code.

LOAD A PROGRAM

The general format of the DFHPC macro instruction to load a program, table, or map from its location in a CICS/VS program library is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHPC │  TYPE=LOAD                                          │
│      │      │ [,PROGRAM=name]                                      │
│      │      │ [,LOADLST=NO]                                        │
│      │      │ [,COND=YES]                                          │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,PGMIDER=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=LOAD
indicates that a program, table, or map is to be loaded into main storage from a CICS/VS program library.

PROGRAM=name
is the identification of the program, table, or map to be loaded; if omitted, the identification is assumed to be in TCAPCPI. TCAPCPI is an eight-character field; identifications less than eight characters must be padded right with blanks.

LOADLST=NO
indicates that the loaded module is not to be deleted when the task issuing the load request is terminated; that is, the loaded module remains resident until deleted at the request of this task or of another task.

COND=YES
indicates that control is to be returned to the program issuing this macro instruction if the requested program cannot be found in the PPT or is disabled. If this operand is omitted and the requested program cannot be found or is disabled, the task is abnormally terminated with an APCT ABEND code.

NORESP and PGMIDER
are used to test the CICS/VS response to this request for program management services. These operands can be specified in this macro instruction or in a DFHPC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Program Services."

RETURN PROGRAM CONTROL

The general format of the DFHPC macro instruction to return control from an application program to the program at the next higher logical level is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHPC │  TYPE=RETURN                                        │
│      │      │ [,TRANSID=transaction code]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=RETURN
indicates that program control is to be returned to a program at the next higher logical level.

TRANSID=transaction code
is the transaction identification to be used with the next input message entered from the terminal with which this requesting task has been associated prior to this request for return of control.

DELETE A LOADED PROGRAM

The general format of the DFHPC macro instruction to delete a previously loaded program is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │DFHPC │  TYPE=DELETE                                         │
│      │      │ [,PROGRAM=name]                                      │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=DELETE
indicates that a previously loaded program, table, or map is no longer required.

PROGRAM=name
is the identification of the program to be deleted; if omitted, the identification is assumed to be in TCAPCPI. TCAPCPI is an eight-character field; identifications less than eight characters must be padded right with blanks.

ABNORMALLY TERMINATE A TRANSACTION

The general format of the DFHPC macro instruction to abnormally terminate a transaction (task) is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │DFHPC │  TYPE=ABEND                                          │
│      │      │  ,ABCODE= value                                      │
│      │      │          YES                                         │
│      │      │ [,CANCEL=YES]                                        │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=ABEND
indicates that a transaction is to be terminated abnormally.

ABCODE
indicates that main storage related to the transaction is to be dumped and provides a four-character abnormal termination code to identify the output dump.

value
is a combination of four alphabetic, numeric, and/or special characters to be printed as the abnormal termination code.

YES
indicates that the abnormal termination code has been placed in TCAPCAC.

Note: If a dump is requested, any information in the common control area of the application program communication section of the TCA is likely to be different in the dump because of the issuance of this macro instruction. If it is necessary to look at information in this area, it should be saved elsewhere prior to issuing this macro instruction.

CANCEL=YES
indicates that all exits established by DFHPC TYPE=SETXIT macro instructions at any level in the task are to be canceled; in effect, they are ignored.

ACTIVATE OR CANCEL AN EXIT FOR ABNORMAL TERMINATION PROCESSING

The general format of the DFHPC macro instruction to activate or cancel an exit to a user-written routine or program to be executed upon abnormal termination of a transaction (task) is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHPC │  TYPE=SETXIT                                        │
│      │      │  ,PROGRAM= name    ,ROUTINE= symbolic address        │
│      │      │            YES               YES                     │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,PGMIDER=symbclic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=SETXIT
indicates that a user exit is to be

1. Activated if the PROGRAM or ROUTINE operand is specified in this DFHPC TYPE=SETXIT macro instruction

2. Canceled if no additional operands are specified in this DFHPC TYPE=SETXIT macro instruction

PROGRAM=
identifies the program to receive control if abnormal termination occurs.

name
is the program name as specified in the processing program table (PPT).

YES
indicates that the name of the program to receive control has been placed in TCAPCPI. TCAPCPI is an eight-character field; identifications less than eight characters must be padded right with blanks.

ROUTINE=
identifies the routine to receive control if abnormal termination occurs. (This parameter applies to Assembler and COBOL programs only.)

symbolic address
is the symbolic address of the routine to receive control.

YES
indicates that the address of the program to receive control has been placed in TCAPCERA.

NORESP and PGMIDER
are used to test the CICS/VS response to this request for program management services. These operands can be specified in this macro instruction or in a DFHPC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Program Services."

REACTIVATE AN EXIT FOR ABNORMAL TERMINATION PROCESSING

The general format of the DFHPC macro instruction to reactivate an exit to a user-written routine or program to be executed upon abnormal termination of a transaction (task) is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────┐
│      │       │                                                  │
│      │ DFHPC │   TYPE=RESETXIT                                  │
│      │       │                                                  │
└──────┴───────┴──────────────────────────────────────────────────┘
```

where:

TYPE=RESETXIT
indicates that an exit to user-written abnormal termination processing is to be reactivated after a preceding application program cancelation or CICS/VS cancelation upon execution of the exit routine.

CONVERT SYMBOLIC LABEL TO ADDRESS

The general format of the DFHPC macro instruction to convert a symbolic label appearing in an American National Standard (ANS) COBOL program to an address is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────┐
│      │       │                                                  │
│      │ DFHPC │   TYPE=COBADDR                                   │
│      │       │   ,LABEL=symbolic label                          │
│      │       │                                                  │
└──────┴───────┴──────────────────────────────────────────────────┘
```

where:

TYPE=COBADDR
indicates that the address of the location represented by a symbolic label is to be returned in TCAPCLA to the application program.

LABEL=symbolic label
is the symbolic label that represents the location in the ANS COBOL program for which the address is required.

TEST RESPONSE TO A REQUEST FOR PROGRAM SERVICES

The general format of the DFHPC macro instruction to test the CICS/VS response to a request for program management services is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHPC │  TYPE=CHECK                                          │
│      │       │  [,NORESP=symbolic address]                          │
│      │       │  [,PGMIDER=symbolic address]                         │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
indicates that the CICS/VS response to a preceding DFHPC macro instruction is to be checked.

NORESP=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if no errors occur during program control processing. NORESP signifies "normal response."

PGMIDER=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the requested program cannot be found in the PPT or is disabled. This operand is applicable to a DFHPC TYPE=LINK or TYPE=LOAD macro instruction in which the COND operand is specified, and to a DFHPC TYPE=SETXIT macro instruction in which the PROGRAM operand is specified.

DFHIC MACRO INSTRUCTION

The interval control macro instruction (DFHIC) is used to request time services as explained below.

TIME-OF-DAY UPDATING

The general format of the DFHIC macro instruction to request updating of CICS/VS-maintained time-of-day values is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHIC │  TYPE=GETIME                                         │
│      │       │ [,FORM={BINARY}]                                     │
│      │       │ [      {PACKED}]                                     │
│      │       │ [,TIMADR={symbolic address}]                         │
│      │       │ [        {YES             }]                         │
│      │       │  [,NORESP=symbolic address]                          │
│      │       │  [,INVREQ=symbolic address]                          │
│      │       │  [,ERROR=symbolic address]                           │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=GETIME
indicates that one or both time-of-day values maintained by CICS/VS are to be updated to the current clock time.

FORM=
indicates which time-of-day representation is desired.

BINARY
specifies that a binary representation of time of day (a four-byte positive value in hundredths of a second) is to be updated and retained in CSACTODB.

PACKED
specifies that the binary representation of time of day (described above) and the packed decimal representation (a four-byte positive value of the form HHMMSSt+ where seconds are truncated to tenths of a second) are to be updated and retained in CSACTODB and CSATODP respectively.

Note: ANS COBOL and PL/I programmers should be aware that the zone portion of the lcw-order byte of this positive number contains hexadecimal F rather than C or D.

TIMADR=
is used when the time of day is to be returned in an application programmer-selected four-byte field. For FORM=BINARY, the binary representation is returned; for FORM=PACKED, the packed decimal representation is returned.

symbolic address
is the symbolic address of the field in which the time of day is to be made available to the application program.

YES
indicates that the symbolic address of the field for the time of day is in TCAICDA.

If this operand is omitted, the fields of the CSA are updated, but the time of day is not placed in another field for reference by the application program.

NORESP, INVREQ, and ERROR
are used to test the CICS/VS response to this request for updating of CICS/VS-maintained time-of-day values. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Time Services" in this chapter.

DELAY PROCESSING OF A TASK

The general format of the DFHIC macro instruction to delay processing of a task until a specified time occurs is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────────┐
│      │      │                                                          │
│      │ DFHIC │ TYPE=WAIT                                               │
│      │      │ [,INTRVAL={numeric value}][,TIME={numeric value}]        │
│      │      │           {YES          }]      {YES          }]        │
│      │      │ [,REQID={name    }]                                      │
│      │      │         {YES     }                                       │
│      │      │         {'prefix'}]                                      │
│      │      │ [,NORESP=symbolic address]                               │
│      │      │ [,INVREQ=symbolic address]                               │
│      │      │ [,EXPIRD=symbolic address]                               │
│      │      │ [,ERROR=symbolic address]                                │
│      │      │                                                          │
└──────┴──────┴──────────────────────────────────────────────────────────┘
```

where:

TYPE=WAIT
indicates that task synchronization, in the form of delay until a specified time occurs, is desired.

INTRVAL=
specifies the interval of time that a task is to be suspended.

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59. This numeric value is added to the current clock time by CICS/VS when the DFHIC TYPE=WAIT instruction is executed to calculate the time of day (clock time) when processing of the task is to be resumed.

YES
indicates that the interval of time (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

If this operand is specified, the TIME operand cannot be specified.

TIME=
specifies the time of day at which processing of the task is to be resumed. If the specified time of day is the same as the current clock time or up to and including six hours preceding the current clock time, the specified time is considered to have occurred and the task is not delayed.

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59.

YES
indicates that the time of day (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

If this operand is specified, the INTRVAL operand cannot be specified.

REQID=
is an optional operand used to assign a unique request identification to this request, as a means of symbolically identifying the request. It should be used if the application programmer wishes to provide another task with the capability of canceling an unexpired WAIT request (see the discussion of DFHIC TYPE=CANCEL). The data is put in temporary storage with this identification.

name
is a unique identifier, up to eight characters in length, selected for this request by the application programmer.

YES
indicates that an eight-character request identification has been placed in TCAICQID by the application program.

'prefix'
is a two-character (including blanks) prefix to be affixed to the Request Identification generated by CICS/VS. If REQID='' is specified, the prefix is assumed to be in the two-byte field TCAICQPX.

If this operand is omitted, CICS/VS generates a unique request identification in the form "DFHNNNNN"; the prefix is DF.

NORESP, INVREQ, EXPIRD, and ERROR
are used to test the CICS/VS response to this request for task synchronization. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Time Services" in this chapter.

SIGNAL EXPIRATION OF A SPECIFIED TIME

The general format of the DFHIC macro instruction to request that CICS/VS indicate when a specified time has expired is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHIC │  TYPE=POST                                          │
│      │      │ [,INTRVAL={numeric value}][,TIME={numeric value}]    │
│      │      │           {YES          }       {YES          }    │
│      │      │ [,REQID={name    }]                                  │
│      │      │         {YES     }                                   │
│      │      │         {'prefix'}                                   │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,INVREQ=symbolic address]                           │
│      │      │ [,EXPIRD=symbolic address]                           │
│      │      │ [,ERROR=symbolic address]                            │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=POST
: indicates that CICS/VS is to make a four-byte timer event control area available to the application program for testing. The area is initialized to binary zeros, and its address is returned in TCAICTEC to the application program. This area is available to the application program for the duration of the task. This area is overridden if the application program issues another DFHIC request of the following types: POST, WAIT, PUT, or INITIATE.

INTRVAL=
: specifies the interval of time that is to elapse before CICS/VS posting is to occur.

  numeric value
  : is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59. This numeric value is added to the current clock time by CICS/VS when the DFHIC TYPE=POST instruction is executed to calculate the time of day (clock time) when posting is to occur.

  YES
  : indicates that the interval of time (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

  If this operand is specified, the TIME operand cannot be specified.

TIME=
: specifies the time of day at which posting is to occur. If the specified time of day is the same as the current clock time or up to and including six hours preceding the current clock time, the specified time is considered to have occurred and posting occurs immediately.

  numeric value
  : is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59.

YES
indicates that the time of day (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

If this operand is specified, the INTRVAL operand cannot be specified.

REQID=
is an optional operand used to assign a unique request identification to this request, as a means of symbolically identifying the request.

name
is a unique identifier, up to eight characters in length, selected for this request by the application programmer.

YES
indicates that a unique eight-character request identification has been placed in TCAICQID by the application program.

'prefix'
is a two-character (including blanks) prefix to be affixed to the Request Identification generated by CICS/VS. If REQID='' is specified, the prefix is assumed to be in the two-byte field TCAICQPX.

If this operand is omitted, CICS/VS generates a unique request identification, which is returned in TCAICQID to the application program; the prefix is DF.

NORESP, INVREQ, EXPIRD, and ERROR
are used to test the CICS/VS response to this request for posting. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Time Services" in this chapter.

INITIATE A TASK

The general format of the DFHIC macro instruction to request that CICS/VS initiate a task at some future time is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────────┐
│      │       │                                                          │
│      │ DFHIC │  TYPE=INITIATE                                           │
│      │       │ [,INTRVAL={numeric value}]|[,TIME={numeric value}]       │
│      │       │           {YES         }]|[      {YES         }]       │
│      │       │ [,REQID={name    }]                                      │
│      │       │         {YES     }                                       │
│      │       │         {'prefix'}]                                      │
│      │       │ [,TRANSID=name]                                          │
│      │       │ [,TRMIDNT={name}]                                        │
│      │       │           {YES }]                                        │
│      │       │ [,NORESP=symbolic address]                               │
│      │       │ [,INVREQ=symbolic address]                               │
│      │       │ [,TRNIDER=symbolic address]                              │
│      │       │ [,TRMIDER=symbolic address]                              │
│      │       │ [,ERROR=symbolic address]                                │
│      │       │                                                          │
└──────┴───────┴──────────────────────────────────────────────────────────┘
```

where:

TYPE=INITIATE
indicates that CICS/VS is to initiate a task at some future time.

INTRVAL=
specifies the interval of time that is to elapse before CICS/VS initiates a task.

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59.

YES
indicates that the interval of time (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

This numeric value is added to the current clock time by CICS/VS when the DFHIC TYPE=INITIATE macro instruction is executed to calculate the time of day (clock time) when the task is to be initiated. If the specified interval is zero or if both the INTRVAL and the TIME operands are omitted, the task is initiated immediately.

If this operand is specified, the TIME operand cannot be specified.

TIME=
specifies the time of day at which CICS/VS is to initiate a task. If the specified time of day is the same as the current clock time or up to and including six hours preceding the current clock time, the specified time is considered to have occurred and the task is initiated immediately.

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59.

YES
indicates that the time of day (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

If this operand is specified, the INTRVAL operand cannot be specified.

REQID=
is an optional operand used to assign a unique request identification to this request, as a means of symbolically identifying the request.

name
is a unique identifier, up to eight characters in length, selected for this request by the application programmer.

YES
indicates that a unique eight-character request identification has been placed in TCAICQID by the application program.

'prefix'
is a two-character (including blanks) prefix to be affixed to the Request Identification generated by CICS/VS. If REQID='' is specified, the prefix is assumed to be in the two-byte field TCAICQPX.

If this operand is omitted, CICS/VS generates a unique request identification, which is returned in TCAICQID to the application program; the prefix is DF.

TRANSID=name
is the symbolic transaction identification of the task to be initiated. If this operand is omitted, the transaction identification is assumed to be in TCAICTI.

TRMIDNT=
is the symbolic terminal identification of the terminal associated with the task to be initiated. This operand is required when the task to be initiated must communicate with a terminal; it should be omitted otherwise.

name
is the symbolic terminal identification of the terminal associated with the task to be initiated.

YES
indicates that the symbolic terminal identification has been placed in TCAICTID.

NORESP, INVREQ, TRNIDER, TRMIDER, and ERROR
are used to test the CICS/VS response to this request for automatic task initiation. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed under "Test Response to a Request for Time Services" in this chapter.

TASK INITIATION WITH DATA

The general format of the DFHIC macro instruction to request automatic task initiation and/or request that data be made available to a task is as follows:

```
        DFHIC   TYPE=PUT
                [,INTRVAL={numeric value}][,TIME={numeric value}]
                          {YES           }       {YES           }
                [,REQID={name    }]
                        {YES     }
                        {'prefix'}
                [,TRANSID=name]
                [,TRMIDNT={name}]
                          {YES }
                [,ICDADDR={symbolic address}]
                          {YES              }
                [,NORESP=symbolic address]
                [,INVREQ=symbolic address]
                [,TRNIDER=symbolic address]
                [,TRMIDER=symbolic address]
                [,IOERROR=symbolic address]
                [,ERROR=symbolic address]
```

where:

TYPE=PUT
indicates that CICS/VS is to initiate a nonterminal-oriented task at some future time and makes one data record available to that task, or provides time-ordered data to be made available to a terminal-oriented task that is to be initiated at some future time.

INTRVAL=
specifies the interval of time that is to elapse before CICS/VS initiates a task.

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59.

YES
indicates that the interval of time (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

This numeric value is added to the current clock time by CICS/VS when the DFHIC TYPE=PUT macro instruction is executed to calculate the time of day (clock time) when the task is to be initiated or the data record made available. If the specified interval is zero or if both the INTRVAL and the TIME operands are omitted, the task is initiated immediately.

If this operand is specified, the TIME operand cannot be specified.

TIME=
specifies the time of day at which CICS/VS is to initiate a task. If the specified time of day is the same as the current clock time or up to and including six hours preceding the current clock time, the specified time is considered to have occurred, and the requested service is provided immediately.

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59.

YES
indicates that the time of day (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

If this operand is specified, the INTRVAL operand cannot be specified.

REQID=
is an optional operand used to assign a unique request identification to this request, as a means of symbolically identifying the request and any data associated with it.

name
is a unique identifier, up to eight characters in length, selected for this request by the application programmer.

YES
indicates that a unique eight-character request identification has been placed in TCAICQID by the application program.

'prefix'
is a two-character (including blanks) prefix to be affixed to the Request Identification generated by CICS/VS. If REQID='' is specified, the prefix is assumed to be in the two-byte field TCAICQPX.

If this operand is omitted, CICS/VS generates a unique request identification, which is returned in TCAICQID to the application program; the prefix is DF.

TRANSID=name
is the symbolic transaction identification of the task to be initiated. If this operand is omitted, the transaction identification is assumed to be in TCAICTI.

TRMIDNT=
is the symbolic terminal identification of the terminal associated with the task to be initiated. This operand is required when the task to be initiated must communicate with a terminal; it should be omitted otherwise.

name
is the symbolic terminal identification of the terminal associated with the task to be initiated.

YES
indicates that the symbolic terminal identification has been placed in TCAICTID.

ICDADDR=
specifies the location of the data to be stored for the task to be initiated at some future time.

symbolic address
is the symbolic address of the storage area containing the data to be made available to the task.

YES
indicates that the symbolic address of the storage area containing the data has been placed in TCAICDA.

If no data is to be passed, DFHIC TYPE=INITIATE rather than DFHIC TYPE=PUT should be used.

NORESP, INVREQ, TRNIDER, TRMIDER, IOERROR, and ERROR
are used to test the CICS/VS response to this request for time services. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Time Services" in this chapter.

RETRIEVE TIME-ORDERED DATA

The general format of the DFHIC macro instruction to retrieve data stored by a DFHIC TYPE=PUT macro instruction (issued by another task prior to initiation of this task) is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │DFHIC │ TYPE=GET                                             │
│      │      │ [,ICDADDR={symbolic address}]                        │
│      │      │ [         {YES             }]                        │
│      │      │ [,RELEASE=NO]                                        │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,INVREQ=symbolic address]                           │
│      │      │ [,NOTFND=symbolic address]                           │
│      │      │ [,ENDDATA=symbolic address]                          │
│      │      │ [,IOERROR=symbolic address]                          │
│      │      │ [,TSINVLD=symbolic address]                          │
│      │      │ [,ERROR=symbolic address]                            │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=GET
indicates that data stored by a preceding DFHIC TYPE=PUT macro instruction issued by another task is to be retrieved.

ICDADDR=
is an optional operand used to specify the location of the storage area selected by the application programmer for the retrieved data.

symbolic address
is the symbolic address of the storage area into which the retrieved data is to be placed.

YES
indicates that the symbolic address of the storage area to be used for the data is in TCAICDA.

If this operand is omitted, CICS/VS acquires a storage area large enough to contain the four-byte length field (LLØØ) and data record. The address of the area is returned in TCAICDA to the application program.

RELEASE=NO
indicates that CICS/VS is not to delete the record from temporary storage after obtaining the record for the application program.

Upon completion of a successful DFHIC TYPE=GET,RELEASE=NO request, CICS/VS places the identification of the temporary-storage record in TCAICQID. This record is then available to the user through the DFHTS macro instruction. Using the DFHTS macro instruction, the user can retrieve or release the record, but the record is not available to any subsequent DFHIC TYPE=GET requests.

NORESP, INVREQ, NOTFND, ENDDATA, IOERROR, TSINVLD, and ERROR
are used to test the CICS/VS response to this request to retrieve data. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Time Services" in this chapter.

CANCEL A REQUEST FOR TIME SERVICES

The general format of the DFHIC macro instruction to cancel a DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE, or DFHIC TYPE=PUT request is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │DFHIC │  TYPE=CANCEL                                     │
│      │      │ [,REQID={name}]                                  │
│      │      │ [       {YES }]                                  │
│      │      │ [,NORESP=symbolic address]                       │
│      │      │ [,INVREQ=symbolic address]                       │
│      │      │ [,NOTFND=symbolic address]                       │
│      │      │ [,ERROR=symbolic address]                        │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=CANCEL
: indicates that a request of one of the following types is to be acted upon as follows:

  1. DFHIC TYPE=WAIT issued by another task (now suspended) is to be treated as though expired.
  2. DFHIC TYPE=POST issued by this task is to be removed from the system.
  3. DFHIC TYPE=POST issued by another task is to be treated as though expired.
  4. DFHIC TYPE=INITIATE is to be removed from the system.
  5. DFHIC TYPE=PUT is to be removed from the system.

REQID=
: is the unique request identification of the request to be canceled. This operand is required for cases 1, 3, 4, and 5 described under TYPE=CANCEL above; it should not be specified when a DFHIC TYPE=POST request is canceled by the task that issued the request.

  name
  : is a unique identifier, up to eight characters in length, that was assigned to the request to be canceled.

  YES
  : indicates that the unique eight-character request identification of the request to be canceled has been placed in TCAICQID by the application program.

NORESP, INVREQ, NOTFND, and ERROR
: are used to test the CICS/VS response to this request for request cancelation. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Time Services" in this chapter.

I/O ERROR RETRY

The general format of the DFHIC macro instruction to retry an operation requested by a DFHIC TYPE=GET macro instruction when an I/O error occurs is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │DFHIC │ TYPE=RETRY                                           │
│      │      │ [,RELEASE=NO]                                        │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,INVREQ=symbolic address]                           │
│      │      │ [,NOTFND=symbolic address]                           │
│      │      │ [,IOERROR=symbolic address]                          │
│      │      │ [,ERROR=symbolic address]                            │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=RETRY
specifies that CICS/VS is to retry the retrieval operation requested by a DFHIC TYPE=GET macro instruction.

RELEASE=NO
indicates that CICS/VS is not to release the record from temporary storage after obtaining the record for the application program.

Upon completion of a successful DFHIC TYPE=GET,RELEASE=NO request, CICS/VS places the identification of the temporary-storage record in TCAICQID. Using this identification, the user can retrieve or release the record from temporary storage trough the DFHTS macro instruction; the record is not available to any subsequent DFHIC get requests.

This operand is valid only for a retry of a DFHIC TYPE=GET request.

NORESP, INVREQ, NOTFND, IOERROR, and ERROR
are used to test the CICS/VS response to this request for a retry of a retrieval operation. These operands can be specified in this macro instruction or in a DFHIC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Time Services" (below).

TEST RESPONSE TO A REQUEST FOR TIME SERVICES

The general format of the DFHIC macro instruction to test the CICS/VS response to a request for time services is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHIC │ TYPE=CHECK                                           │
│      │       │ [,NORESP=symbolic address]                           │
│      │       │ [,INVREQ=symbolic address]                           │
│      │       │ [,EXPIRD=symbolic address]                           │
│      │       │ [,TRNIDER=symbolic address]                          │
│      │       │ [,TRMIDER=symbolic address]                          │
│      │       │ [,NOTFND=symbolic address]                           │
│      │       │ [,ENDDATA=symbolic address]                          │
│      │       │ [,IOERROR=symbolic address]                          │
│      │       │ [,TSINVLD=symbolic address]                          │
│      │       │ [,ERROR=symbolic address]                            │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
indicates that the CICS/VS response to a preceding DFHIC macro instruction is to be checked.

NORESP=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if no error occurs. NORESP signifies "normal response."

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an invalid type of request was received for processing by the interval control program.

EXPIRD=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the time specified in a DFHIC TYPE=POST or DFHIC TYPE=WAIT request has expired at the time the request is issued.

TRNIDER=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the symbolic transaction identification specified in a DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request cannot be found in the program control table.

TRMIDER=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the symbolic terminal identification specified in the DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request cannot be found in the terminal control table (TCT).

NOTFND=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the request identification specified in a DFHIC TYPE=CANCEL macro instruction fails to match an unexpired time-ordered request. It is also applicable to DFHIC TYPE=GET or DFHIC TYPE=RETRY requests and signifies that the time-ordered data stored for retrieval through the DFHIC TYPE=PUT macro instruction cannot be located using the unique request identification contained in TCAICQID at the time of this request. This condition occurs on a retrieval operation if some prior

task retrieved the data stored under the request identification directly through temporary storage facilities and then released the data area. It also occurs if the request identification associated with the original DFHIC TYPE=PUT request fails to remain a unique identification.

ENDDATA=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if no more data is stored for the task issuing a DFHIC TYPE=GET request. It can be considered a normal end-of-file response when retrieving sequential time-ordered data records.

IOERROR=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if an input/output error occurs during a DFHIC TYPE=GET or DFHIC TYPE=PUT operation on auxiliary storage. The DFHIC TYPE=RETRY macro instruction can be used in the routine for handling DFHIC TYPE=GET input/output errors.

TSINVLD=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if the CICS/VS temporary storage program does not support a DFHTS TYPE=GET request issued by the CICS/VS interval control program. This situation can occur when a dummy temporary storage program is included in the current CICS/VS system in place of a functional temporary storage program.

ERROR=symbolic address
specifies the entry label of the user-written routines to which control is to be passed if any of the response conditions other than NORESP occurs.

DFHKC MACRO INSTRUCTION

The task control macro instruction (DFHKC) is used to request the following services, as explained below:

1. Initiate a task (ATTACH)
2. Reschedule a 3650 task
3. Change the priority of a task (CHAP)
4. Synchronize a task (WAIT)
5. Synchronize the use of a resource by a task (ENQ and DEQ)
6. Purge a task on system overload (PURGE and NOPURGE)

DFHKC

INITIATE A TASK

The general format of the DFHKC macro instruction to initiate a task is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHKC │ TYPE=ATTACH                                          │
│      │       │ [,FCADDR=symbolic address]                           │
│      │       │ [,TRANSID=name]                                      │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=ATTACH
indicates that a task is to be initiated.

FCADDR=symbolic address
is the symbolic address of the facility control area (FCA) associated with this task; if omitted, the address is assumed to be in TCAKCFA.

TRANSID=name
is the transaction identification for the task; if omitted, the transaction identification is assumed to be in TCAKCTI.

RESCHEDULE A 3650 TASK

The general format of the DFHKC macro instruction to reschedule a 3650 task is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHKC │ TYPE=SCHEDULE                                       │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=SCHEDULE
indicates that a 3650 task, rejected because 3651 resources were not available to load the 3650 application program, is to be rescheduled later.

Refer to the CICS/VS Advanced Communication Guide for 3650 programming considerations.

CHANGE PRIORITY OF A TASK

The general format of the DFHKC macro instruction to change the dispatching priority of a task is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHKC │ TYPE=CHAP                                           │
│      │      │ [,PRTY=priority value]                               │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=CHAP
indicates that the dispatching priority of a task is to be changed.

PRTY=priority value
is a decimal numeral in the range from 0 through 255 to be taken as the priority value for this task; if omitted, the priority value is assumed to be in TCATCDP.

SYNCHRONIZE A TASK

The general format of the DFHKC macro instruction to synchronize the execution of a task with the completion of an event, or to

voluntarily relinquish control to a task of higher priority, is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHKC │ TYPE=WAIT                                           │
│      │      │ ,DCI={SINGLE}                                       │
│      │      │      {LIST  }                                       │
│      │      │      {DISP  }                                       │
│      │      │ [,ECADDR=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=WAIT
indicates that execution of the task is to be synchronized.

DCI=
specifies the circumstances under which synchronization is to occur.

SINGLE
indicates that the task is to be synchronized with the completion of a single event.

LIST
indicates that the task is to be synchronized with the completion of one event in a list of events.

DISP
indicates that the task wishes to give up control to any higher priority task that is ready to be processed; if none exists, control is to be returned to this task.

ECADDR=symbolic address
is used with DCI=SINGLE or DCI=LIST to specify the symbolic address of the single event control area or list of event control areas identifying the event with which this task is to be synchronized; if omitted when SINGLE or LIST is specified, the address is assumed to be in TCATCEA.

ENQUEUE UPON A RESOURCE

The general format of the DFHKC macro instruction to enqueue upon a resource, causing execution of a task to be synchronized with the availability of that resource, is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHKC │ TYPE=ENQ                                        │
│      │      │ [,QARGADR=symbolic address]                      │
│      │      │ [,QARGLNG=number]                                │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=ENQ
indicates that execution of this task is to be synchronized with the availability of a specific resource.

QARGADR=symbolic address
is either the symbolic address of the resource to be enqueued, or the symbolic address of a location that contains a unique argument (for example, an employee name) that represents the resource. If this operand is omitted, the address is assumed to be in the three low-order bytes of TCATCQA, a four-byte field.

QARGLNG=number
is the length, in bytes, of the resource to be enqueued upon. This operand is needed only if the QARGADR operand is a unique argument that represents the resource to be enqueued. If omitted in such a case, the contents of the high-order byte of TCATCQA are assumed to be the length of the argument.

DEQUEUE UPON A RESOURCE

The general format of the DFHKC macro instruction to dequeue upon a resource (effectively, to revoke a preceding enqueue request upon that resource) is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHKC │ TYPE=DEQ                                             │
│      │       │ [,QARGADR=symbolic address]                          │
│      │       │ [,QARGLNG=number]                                    │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=DEQ
: indicates that a preceding enqueue request upon a specific resource is to be revoked.

QARGADR=symbolic address
: is either the symbolic address of the resource to be dequeued, or the symbolic address of a location that contains a unique argument (for example, an employee name) that represents the resource. If this operand is omitted, the address is assumed to be in the three low-order bytes of TCATCQA, a four-byte field.

QARGLNG=number
: is the length, in bytes, of the resource to be dequeued. This operand is needed only if the QARGADR operand is a unique argument that represents the resource to be dequeued. If omitted in such a case, the contents of the high-order byte of TCATCQA are assumed to be the length of the argument.

  If the QARGLNG operand is specified in a DFHKC TYPE=ENQ macro instruction enqueuing upon a resource, the QARGLNG operand must also be specified in the DFHKC TYPE=DEQ macro instruction for that resource, and the values of these operands must be the same.

DECLARE A TASK TO BE PURGEABLE

The general format of the DFHKC macro instruction to declare that a task may be purged if a system stall condition occurs is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHKC │ TYPE=PURGE                                           │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=PURGE
: indicates that the task issuing this instruction may be purged from the system if a system stall condition occurs. (See programming note 1 below for the relationship between DFHKC TYPE=PURGE and the SPURGE option of the DFHPCT TYPE=ENTRY macro.

DECLARE A TASK TO BE NONPURGEABLE

The general format of the DFHKC macro instruction to declare that a task cannot be purged if a system stall condition occurs is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHKC │ TYPE=NOPURGE                                    │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=NOPURGE
indicates that the task issuing this macro instruction cannot be purged from the system if a system stall condition occurs. (See programming note 1 below for the relationship between DFHKC TYPE=NOPURGE and the SPURGE option of the DFHPCT TYPE=ENTRY macro.)

Programming Note:

1. The PURGE and NOPURGE options of the DFHKC macro are intended to be used as temporary overrides to the SPURGE specification in the DFHPCT TYPE=ENTRY macro for a task. For example, if a DFHKC TYPE=NOPURGE macro is issued in a program for a task, the task cannot be purged even though SPURGE=YES is specified in the DFHPCT TYPE=ENTRY macro for the task at system generation. Refer to the publication CICS/VS System Programmer's Reference Manual for details on the SPURGE option of the DFHPCT TYPE=ENTRY macro.

DFHJC MACRO INSTRUCTION

The application programmer requests journal services by issuing DFHJC macro instructions. The formats of the various types of journal requests are explained below.

ACQUIRE A JOURNAL CONTROL AREA (JCA)

The general format of the DFHJC macro instruction to acquire a journal control area (JCA) is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │ DFHJC │ TYPE=GETJCA                                     │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=GETJCA
indicates that a communication area to be used for communication between the application program and the CICS/VS journal control program is to be acquired. The address of the JCA is returned in TCAJCAAD to the application program.

CREATE A JOURNAL RECORD AND WAIT FOR OUTPUT

The general format of the DFHJC macro instruction to create a journal record, initiate its output, and wait for completion is as follows:

```
DFHJC  TYPE={PUT            }
       {(WRITE,WAIT)}
       ,JFILEID={nn     }
                {SYSTEM }
                {YES    }
       [,JTYPEID={nnnn}]
                 {YES }
       [,JCDADDR={symbolic address}]
                 {YES              }
       [,JCDLGTH={decimal value}]
                 {YES           }
       [,PFXADDR={symbolic address}]
                 {YES              }
       [,PFXLGTH={decimal value}]
                 {YES           }
       [,NORESP=symbolic address]
       [,IDERROR=symbolic address]
       [,LERROR=symbolic address]
       [,IOERROR=symbolic address]
       [,NOTOPEN=symbolic address]
       [,INVREQ=symbolic address]
```

where:

TYPE=
: indicates the journal operations required.

  PUT
  : indicates that a journal record is to be created in the journal buffer area and then written out; the requesting task will wait until the physical record has been written.

  (WRITE,WAIT)
  : implies, and is equivalent to, TYPE=PUT.

JFILEID
: is the one-byte identification of the journal file (data set) referred to in this journal operation.

  nn
  : is a decimal value in the range from 2 through 99 to be taken as the journal file identification.

  SYSTEM
  : indicates that the system log data set is the journal for this operation.

  YES
  : indicates that the journal file identification has been placed in JCAJFID prior to issuing this macro instruction.

JTYPEID=
: is an identifier to be placed in the journal record to identify its origin.

nnnn
is a one- to four-character hexadecimal value to be taken as the identifier for the journal record; if fewer than four characters are specified, padding with zeros occurs on the right.

YES
indicates that the journal record identification has been placed in JCAJRTID prior to issuing this macro instruction.

JCDADDR
is the address of the user data to be built into the journal record.

symbolic address
is the symbolic address of the user data.

YES
indicates that the address of the user data has been placed in JCAADATA prior to issuing this macro instruction.

JCDLGTH=
is the length of the user data to be built into the journal record. (See programming note 1 below for the relationship between JCDLGTH and PFXLGTH.)

decimal value
is a decimal numeral in the range from 1 to 32000 (or a lower maximum, because of the journal buffer size), indicating the length, in bytes, of the user data.

YES
indicates that the length, in binary, of the user data has been placed in JCALDATA prior to issuing this macro instruction.

PFXADDR=
is the address of user prefix data to be included in the journal record.

symbolic address
is the symbolic address of the user prefix data.

YES
indicates that the address of the user prefix data has been placed in JCAAPRFX prior to issuing this macro instruction.

DFHJC

PFXLGTH=
is the length of the user prefix data to be included in the journal record. (See programming note 1 below for the relationship between PFXLGTH and JCDLGTH.)

decimal value
is a decimal numeral in the range from 1 to 32000 (or a lower maximum, because of the journal buffer size), indicating the length, in bytes, of the user prefix data.

YES
indicates that the length, in binary, of the user prefix data has been placed in JCALPRFX prior to issuing this macro instruction.

NORESP, IDERROR, LERROR, IOERROR, NOTOPEN, and INVREQ
are used to test the CICS/VS response to this request for journal services. These operands can be specified in this macro

instruction or in a DFHJC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Journal Services" in this chapter.

Programming Note:

1. Because the maximum buffer length that can be used to write a journal record is 32,767 bytes, the combined length specified by JCDLGTH and PFXLGTH (or stored in JCALDATA and JCALPRFX, respectively) cannot exceed 32,767.

CREATE A JOURNAL RECORD

The general format of the DFHJC macro instruction to create a journal record for subsequent output is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │DFHJC │TYPE=WRITE                                            │
│      │      │,JFILEID={nn      }                                   │
│      │      │         {SYSTEM  }                                   │
│      │      │         {YES     }                                   │
│      │      │[,JTYPEID={nnnn}]                                     │
│      │      │         {YES }                                       │
│      │      │[,JCDADDR={symbolic address}]                         │
│      │      │         {YES              }                          │
│      │      │[,JCDLGTH={decimal value}]                            │
│      │      │         {YES           }                             │
│      │      │[,PFXADDR={symbolic address}]                         │
│      │      │         {YES              }                          │
│      │      │[,PFXLGTH={decimal value}]                            │
│      │      │         {YES           }                             │
│      │      │[,STARTIO={YES}]                                      │
│      │      │         {NO }                                        │
│      │      │[,COND={(YES,symbolic address)}]                      │
│      │      │       {NO                    }                       │
│      │      │[,NORESP=symbolic address]                            │
│      │      │[,IDERROR=symbolic address]                           │
│      │      │[,LERROR=symbolic address]                            │
│      │      │[,NOTOPEN=symbolic address]                           │
│      │      │[,INVREQ=symbolic address]                            │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=WRITE
: indicates that a journal record is to be created in the journal's buffer area, and control is to be returned to the requesting task immediately. (See programming note 1.)

JFILEID, JTYPEID, JCDADDR, JCDLGTH, PFXADDR, and PFXLGTH
: are used in this macro instruction as in the DFHJC macro instruction described in the section entitled "Create a Journal Record and Wait for Output," which immediately precedes this discussion.

STARTIO=
: specifies whether output of the journal record is to be initiated immediately.

  YES
  : indicates that output of the journal record is to be initiated.

NO
indicates that no output operation is required at this time.

COND=
specifies whether control is to be returned to the application program if the request cannot be satisfied immediately because insufficient journal buffer space is available. If control is to be returned, the point of return must be specified as a second parameter of this operand.

(YES,symbolic address)
indicates that control is to be returned to the location represented by symbolic address in the application program if the request cannot be satisfied immediately. No journal record will have been created for the request.

NO
indicates that the contents of the current buffer are to be written out and the requesting task placed in a wait state until its request has been satisfied (by the building of a record in buffer space freed by the write operation).

NORESP, IDERROR, LERROR, NOTOPEN, and INVREQ
are used to test the CICS/VS response to this request for journal services. These operands can be specified in this macro instruction or in a DFHJC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Journal Services" in this chapter.

Programming Note:

1. At some later time, the task may wish to ensure that the journal record has been written. If the JCA is to be used for any other journal requests, that task should save the event control number (four bytes) returned in JCAECN after a journal record is successfully created in response to the DFHJC TYPE=WRITE request. The event control number must be restored to the JCA immediately before the DFHJC TYPE=WAIT request used to check and wait for output. If the JCA is not used in the interim for any other journal requests for the task, there is no need to save and restore the event control number.

   However, restoring the event control number prior to issuing a DFHJC TYPE=WAIT macro is a good programming practice. CICS/VS management modules also use the JCA of the task for journal requests. For example, automatic journaling is used in the file control program, and logging can be performed for recovery purposes at the user's option.

WAIT FOR OUTPUT OF A JOURNAL RECORD

The general format of the DFHJC macro instruction to wait for output of a previously created journal record is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHJC │ TYPE=WAIT                                           │
│      │      │ ,JFILEID={nn      }                                  │
│      │      │          {SYSTEM  }                                  │
│      │      │          {YES     }                                  │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,IDERROR=symbolic address]                          │
│      │      │ [,IOERROR=symbolic address]                          │
│      │      │ [,NOTOPEN=symbolic address]                          │
│      │      │ [,INVREQ=symbolic address]                           │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=WAIT
: indicates that the requesting task is to be placed in a wait state until the block containing a journal record has been written as output (that is, the journal operation is to be synchronized with continued execution of the task issuing the journal write request). If the block containing the journal record has not been written, the physical write is initiated and the requesting task is placed in a wait state until the write is completed (see programming note 1 under "Create a Journal Record" above).

JFILEID
: is the one-byte identification of the journal file (data set) referenced in this journal operation.

  nn
  : is a decimal value in the range from 2 through 99 to be taken as the journal file identification.

  SYSTEM
  : indicates that the system log data set is the journal for this operation.

  YES
  : indicates that the journal file identification has been placed in JCAJFID prior to issuing this macro instruction. The operand can be specified if the user has restored the event control number (see programming note 1 under "Create a Journal Record" above), since JCAJFID is part of the event control number.

NORESP, IDERROR, IOERROR, NOTOPEN, and INVREQ
: are used to test the CICS/VS response to this request for journal services. These operands can be specified in this macro instruction or in a DFHJC TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for Journal Services" (below).

TEST RESPONSE TO A REQUEST FOR JOURNAL SERVICES

The general format of the DFHJC macro instruction to check the CICS/VS response to a request for journal services is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHJC │ TYPE=CHECK                                           │
│      │       │ [,NORESP=symbolic address]                           │
│      │       │ [,IDERROR=symbolic address]                          │
│      │       │ [,LERROR=symbolic address]                           │
│      │       │ [,IOERROR=symbolic address]                          │
│      │       │ [,NOTOPEN=symbolic address]                          │
│      │       │ [,INVREQ=symbolic address]                           │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
indicates that the CICS/VS response to a preceding DFHJC TYPE=PUT, TYPE=WRITE, TYPE=WAIT, or TYPE=(WRITE,WAIT) request is to be checked.

NORESP=symbolic address
is the address to which control is to be returned if the requested operation was performed successfully.

IDERROR=symbolic address
is the address to which control is to be returned if the specified journal file identification does not exist in the journal control table (JCT).

LERROR=symbolic address
is the address to which control is to be returned if the computed length for the journal record exceeds the total buffer space allocated for the journal data set, as specified in the JCT entry for the data set.

IOERROR=symbolic address
is the address to which control is to be returned if the physical output of a journal record was not accomplished because of an unrecoverable I/O error. This operand is applicable only to requests that may cause a wait for completion of output, that is, to TYPE=PUT, TYPE=(WRITE,WAIT), or TYPE=WAIT.

NOTOPEN=symbolic address
is the address to which control is to be returned if the journal request cannot be satisfied because the specified journal data set has been disabled and is not available.

INVREQ=symbolic address
is the address to which control is to be returned if the TYPE operand is invalid.

DFHSP

DFHSP MACRO INSTRUCTION

The application programmer uses the sync point macro instruction to divide a task (usually, a long-running one) into smaller units, known as logical units of work. Each sync point macro instruction indicates that a logical unit of work is to be completed and that all resources owned by the task up to this point are to be released (dequeued).

The format of the sync point macro instruction is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHSP │ TYPE=USER                                           │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

TYPE=USER
: causes an application-program-defined sync point to be established, marking the completion of a logical unit of work.

CHAPTER 8. PROGRAM TESTING AND DEBUGGING

CICS/VS user-written application programs execute in an interactive, information system environment. Testing programs in such an environment has always been difficult. The information system, including the operating system, CICS/VS, and application programs, must be responsive currently to many factors. The equipment configuration includes many lines and terminals through which requests for varied services are entered on a random, nonscheduled basis. The precise relationship of all programs and data set (file) activity generated from the terminal inputs differs from one moment to the next.

Because of the complexity of this situation, numerous aids to program testing and debugging have been included within CICS/VS. Sequential terminal support is provided primarily to serve as a tool in the testing environment. The CICS/VS system monitoring component (introduced in the first section of this manual) comprises two functions which provide important services to application programs:

- Trace Management - Provides a trace table containing entries that reflect the execution of various CICS/VS macro instructions by user-written application programs and CICS/VS management functions.

- Dump Management - Provides a facility to assist in analysis of programs and transactions undergoing development or modification. Specified areas of main storage are dumped onto a sequential data set, either tape or disk, for subsequent offline formatting and printing using a CICS/VS utility program.

Testing of programs and monitoring of activities are continuous processes. Neither is completed when a given program or set of programs is up and running smoothly. As existing programs are upgraded, and new applications are added, testing and monitoring cycles must be repeated to ensure that the complete system continues to function in an uninterrupted, efficient manner.

SEQUENTIAL TERMINAL SUPPORT

Even at the simplest level of program testing, the implementer faces problems. It is not efficient to test a program from a terminal if all test data must be keyed into the system from that terminal for each test shot. The programmer cannot easily retain a backlog of proven test data and quickly test programs through the key-driven terminal as changes are made.

CICS/VS allows the application programmer to begin testing his programs without the use of a telecommunication device. It is possible to specify through the terminal control table that sequential devices be used as terminals. These sequential devices may be card readers, line printers, disk units, or magnetic tape units. In fact, a terminal control table can include combinations of sequential devices such as: card reader and line printer, one or more disk or tape data sets as input, one or more disk or tape data sets as output. A table that contains references to these card-reader-in-line-printer-out (CRLP) terminals can also include references to other terminals on the system.

The input data submitted from a sequential device must be prepared in the form that it would come from a telecommunications device. A one- to four-character transaction identification only, or if data is included, a one- to four-character transaction identification (followed

by a system-defined transaction code delimiter or a blank it less than four) must appear in the first one to four positions of the first input for a transaction. If a sequential device is being used as a terminal, an end-of-data indicator, a 0-2-8 punched card code (X'E0') or the equivalent as specified at system generation, must follow the input message or the system-defined data termination character. The input is processed sequentially and must be unblocked. The Sequential Access Method (SAM) is used to read and write the necessary inputs and outputs. The operating system utilities can be used to create the input data sets and print the output data sets.

Using this approach, it is possible to prepare a stream of transaction test cases to do the basic testing of a program module. As the testing progresses, the user can generate additional transaction streams to validate the multiprogramming capabilities of his programs or to allow transaction test cases to be run concurrently.

At some point in testing, it is necessary to use telecommunication devices to ensure that the transaction formats are satisfactory, the terminal operational approach is satisfactory, and the transactions can be processed on the terminal. The terminal control table can be altered to contain more and different devices as the testing requirements change.

When the testing has proven that multiple transactions can be processed concurrently and the necessary data sets (actual or duplicate) for online operation have been created, the user begins testing in a controlled environment with the telecommunication devices. In this controlled environment, the transaction test cases should represent all functions of the eventual system, but on a smaller, measurable scale. For example, a company whose information system will work with 15 district offices may select one district office for the controlled test. During the controlled test, all transactions, data set activity, and output activity from the system should be measured closely.

Requests for input or output from a sequential terminal are expressed by means of terminal control macro instructions (DFHTC), just as other requests for input/output operations.

In response to a DFHTC TYPE=READ, where the terminal has been described in the terminal control table as a CRLP, DISK, or TAPE terminal, data is read from the input data set until any of the following occurs:

- An end-of-data indicator is detected in the input stream. (The indicator must be defined by the user at system generation time.)
- Sufficient input has been read to fill the input area associated with the line used for transmission. If an end-of-data indicator is not detected before the input area is filled, all further data preceding an end-of-data indicator is bypassed and treated as a system error, which is passed to the user-installation terminal error program (DFHTEP).
- End of file (EOF) is detected. The READ is considered complete. Any subsequent READ is treated as a system error, which is passed to the user-installation terminal error program (DFHTEP) with a response code of 4. (Under CICS/DOS/VS, EOF applies to a card reader only.)

In response to a DFHTC TYPE=WRITE from a CRLP terminal, multiple lines are written in print format as follows:

- If there is no new-line (X'15') character within the number of characters contained in one print line of the specified line size

(as found in TCTTELPL, a field in the TCTTE), the output is written in fixed-length lines of the size specified.

- If new-line characters are encountered, a new line is begun for each one. Writing of output continues until the end of the terminal input/output area (TIOA) is reached.

For additional information concerning DFHTC macro instructions, see "Terminal Services" and "DFHTC Macro Instruction" in this manual.

TRACE SERVICES

The CICS/VS trace facility is designed as a debugging aid for application programmers and IBM Field Engineering. This facility makes use of a trace table consisting of standard and nonstandard entries produced by requests for trace control services. In a debugging environment, the application programmer can locate the trace table in primary storage. Register 13 points to the beginning of the CSA. The address of the first entry in the trace table header is available at the contents of register 13 plus X'11C' (the CSA field identified by the symbolic label CSTRTBA). As explained under "Trace Table" in this chapter. The first three words of the trace table contain the sub header HEADER AT and the fourth word contains the address of the trace table header. The first word in the trace table header is the address of the last-used entry, which the application programmer can refer to, to determine what was happening when a problem occurred. The second word contains the address of the start of the trace table itself. Because entries are made to the trace table in order of occurrence, the programmer can work backward in the trace table to determine prior activity. The meanings of trace table entries are explained in detail under "Trace Table."

If the CICS/VS auxiliary trace is active, trace entries written to the Trace Table, which is in main storage, are also written to the auxiliary-trace data set. Trace entries written to this data set do not wrap around as do those in the trace table. The CICS/VS trace utility program can be used to process and print the trace records written to the trace data set. This utility can be used to print all entries on the data set or only selected entries.

Standard entries are recorded in the CICS/VS trace table each time one of the following CICS/VS macro instructions is issued by an application program or by a CICS/VS management or service program:

- DFHKC (Task Control)
- DFHSC (Storage Control)
- DFHPC (Program Control)
- DFHIC (Interval Control)
- DFHDC (Dump Control)
- DFHFC (File Control)
- DFHTD (Transient Data Control)
- DFHTS (Temporary Storage Control)
- DFHJC (Journal Control)
- DFHBMS (Basic Mapping Support)
- DFHBIF (Built-In Functions)
- DFHTC (Terminal Control for VTAM-Supported Terminals Only)
- CICS/VS-DL/I Interface

Each standard entry contains a unique trace identification from 240 through 255 (X'F0' through X'FF') and information to aid the application programmer in determining where the macro instruction was issued and what type of request was made to the management program. The

application programmer need code no additional instructions to use this tool as an aid in the debugging process.

The trace identification numbers 200 through 229 (X'C8' through X'E5') are reserved for CICS/VS system trace entries. The application programmer can make direct, nonstandard entries in the trace table by using the DFHTR macro instruction. A trace identification number from 0 through 199 (X'00' through X'C7') and accompanying data is assigned for each trace entry. Thus, by defining several unique trace entries, the programmer can trace the logical path through a particular application or group of application programs.

An additional trace function, the field engineering (FE) class trace, is normally inhibited but may be activated by the user or IBM Field Engineering for debugging. The trace identification numbers 230 through 239 (X'E6' through X'EF') have been set aside for this purpose. Entries in this class are produced by the terminal abnormal condition program, provided that such entries have been requested by means of a DFHTR macro instruction in the application program. X'E6' is documented in the trace table below. X'E7' through X'EF' are reserved for field engineering use.

Trace control is branched to by the requesting program and executes as a service routine under the RCA of the requesting program. Registers are saved and restored. Return after the requested service has been performed is to the next sequential instruction in the requesting program.

The trace control macro instruction (DFHTR) is used to request any of the following services:

1. Dynamically allow the trace facility to begin logging appropriate entries into the trace table.

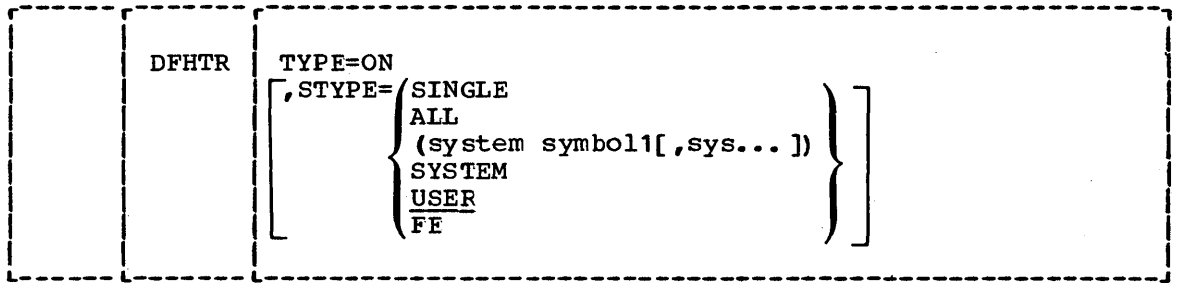
2. Dynamically cause the trace facility to stop logging entries into the trace table.

3. Dynamically cause a specified entry to be logged into the trace table.

The general format of each trace table entry is described under "trace table" in this section. A detailed explanation of trace table entries for CICS/VS management programs is provided. The application programmer should refer to the introductory description of the general format before reading the discussion of the DFHTR macro instruction, which follows.

TRACE ON FUNCTION

The ON function of trace control is used to dynamically allow the trace facility to begin logging appropriate entries into the trace table. The application programmer invokes it by use of the DFHTR TYPE=ON macro instruction. The format of this macro instruction is as follows:



where:

TYPE=ON
initiates the logging function of the trace facility.

STYPE=

indicates the type of entries to be logged.

SINGLE
specifies that the trace capability is to be turned on for user entries of the single transaction issuing the DFHTR macro instruction for the duration of the task.

ALL
specifies that all tracing facilities are to be activated.

(system symbol1[,sys...])
specifies one or more system symbols to selectively turn on appropriate system macro trace facilities. The valid system symbols are as follows:

Symbol	Meaning
KC	Task Control (DFHKC)
SC	Storage Control (DFHSC)
PC	Program Control (DFHPC)
IC	Interval Control (DFHIC)
DC	Dump Control (DFHDC)
FC	File Control (DFHFC)
TD	Transient Data Control (DFHTD)
TS	Temporary Storage Control (DFHTS)
JC	Journal Control (DFHJC)
BM	Basic Mapping Support (DFHBMS)
BF	Built-In Functions (DFHBIF)
TC	Terminal Control (DFHTC)

SYSTEM
specifies that the trace capability is to be turned on for CICS/VS. This operand turns on the system master trace flag which must be on in addition to the individual system trace flags before tracing of any system macros occurs.

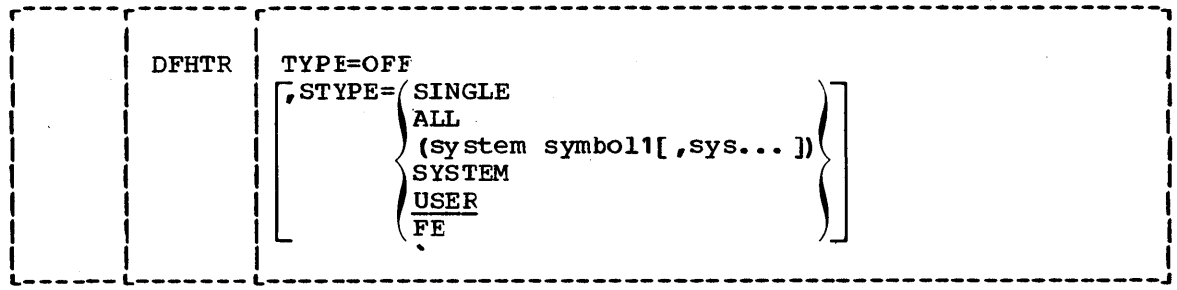
USER
specifies that the trace capability is to be turned on for all user entries for all current transactions; that is, causes the trace facility to begin logging user entries to the trace table for all transactions currently active in the system.

FE
specifies that the trace capability is to be turned on for all Field Engineering (FE) entries. This parameter is valid for only Assembler-language programs.

If this operand is omitted, USER is assumed.

TRACE OFF FUNCTION

The OFF function of trace control is used to dynamically cause the trace facility to stop logging entries into the trace table. The application programmer invokes this function by issuing the DFHTR TYPE=OFF macro instruction. The format of this macro instruction is as follows:



where:

TYPE=OFF
indicates that the logging of certain types of entries into the trace table is to be stopped.

STYPE=
indicates the type of entries for which logging is to be discontinued.

SINGLE
specifies that the tracing of user entries is to be terminated for the single transaction issuing this macro instruction.

Note: This is the only parameter that overrides a preceding DFHTR TYPE=ON,STYPE=SINGLE macro instruction issued by this transaction.

ALL
specifies that all tracing facilities are to be deactivated.

(system symbol1[,sys...])
indicates the system macro trace facilities to be deactivated in the same manner as in the DFHTR TYPE=ON macro instruction above.

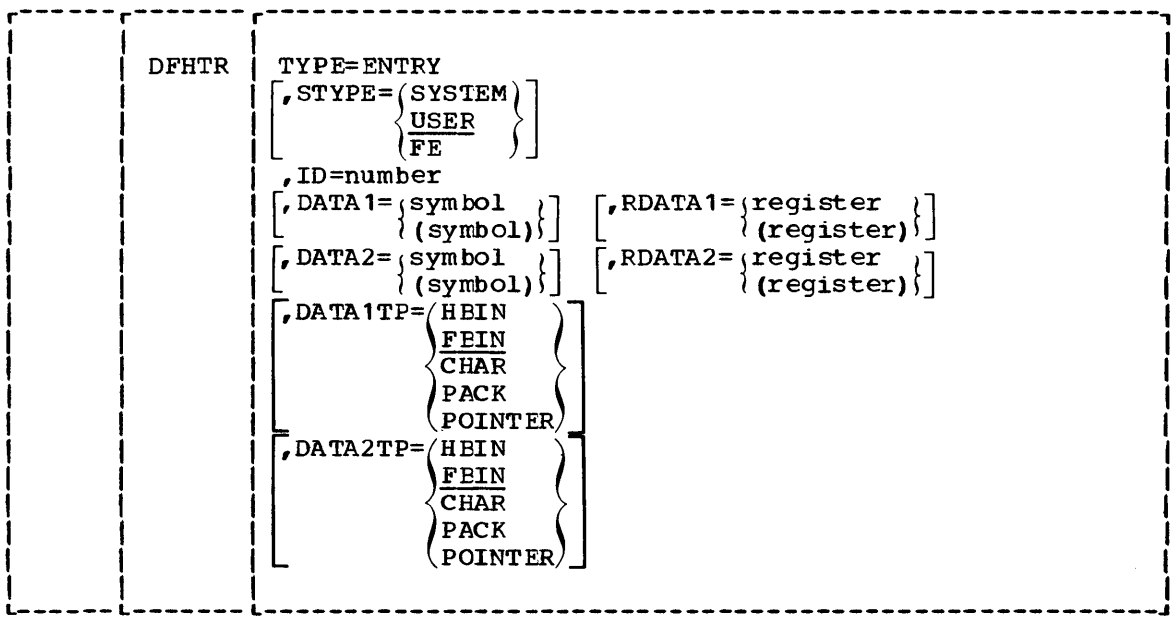
SYSTEM
specifies that the trace capability is to be turned off for all entries made from within CICS/VS.

USER
specifies that the trace capability is to be turned off for all user entries for those transactions for which the capability was invoked by means of a DFHTR TYPE=ON,STYPE=USER macro instruction.

FE
specifies that the tracing of Field Engineering (FE) entries is to be terminated for all transactions. This parameter is valid for only Assembler-language programs.

TRACE ENTRY FUNCTION

The ENTRY function of trace control is used to dynamically cause a specified entry to be logged into the trace table if the trace facility has been turned on for that type of entry. The application programmer invokes this function by issuing the DFHTR TYPE=ENTRY macro instruction. The format of this macro instruction is as follows:



where:

TYPE=ENTRY
indicates that logging of a particular entry to the trace table is required at this point.

STYPE=
indicates the type of the entry to be logged.

SYSTEM
identifies the entry as a CICS/VS entry.

USER
identifies the entry as an application program entry.

FE
identifies the entry as a Field Engineering entry. This parameter is valid for only Assembler-language programs.

ID=number
specifies the trace identification number for this entry (byte 0 of the trace table entry) and must be coded as a self-defining term. A value from 0 through 199 may be specified when STYPE=USER; a value from 200 through 229 may be specified when STYPE=SYSTEM; and a value from 230 through 239 may be specified when STYPE=FE. (The trace identification number 230 is included in all entries produced by the terminal abnormal condition program.) Numbers 240 through 253 (X'F0' through X'FD') are reserved for system macro instruction trace entries. The numbers 254 (X'FE') and 255 (X'FF') indicate TYPE=ON and TYPE=OFF entries, respectively.

DATA1=
specifies the address of the data to be placed in the first data field (bytes 8 to 11) of the trace table entry.

symbol
is the symbolic address of the data to be placed in the first data field of the table entry.

(symbol)
is the symbolic address of an area that contains the address of the data to be placed in the first data field.

When this macro instruction is issued in a high-level language program, if DATA1 is specified, DATA1TP is required.

RDATA1=
specifies the register whose contents are to be placed in the first data field of the trace table entry. This parameter is valid for Assembler language only.

register
the number of the register whose contents are to be placed in the first data field of the table entry.

(register)
the number of the register whose contents are the address of the data to be placed in the first data field.

DATA2=
is similar to DATA1 except that it is used for the second data field (bytes 12 to 15) of the trace table entry.

When this macro instruction is issued in a high-level language program, if DATA2 is specified, DATA2TP is required.

RDATA2=
is similar to RDATA1 except that it is used for the second data field of the trace table entry. This parameter is valid for Assembler language only.

DATA1TP=
specifies the format of the data to be placed in the first data field of the trace table entry. The meanings of the keyword parameters are as stated below:

Specification	Data Format	Field Definition
DATA1TP=HBIN	Halfword, binary	COBOL: 9(4) COMP PL/I: BIN FIXED(15)
DATA1TP=FBIN	Fullword, binary	COBOL: 9(8) COMP PL/I: BIN FIXED(31)
DATA1TP=CHAR	1 to 4 characters	COBOL: X(4) PL/I: CHAR(4)
DATA1TP=PACK	1 to 4 bytes, packed decimal	COBOL: 9(7) COMP-3 PL/I: DEC FIXED(7)
DATA1TP=POINTER	PL/I pointer variable	PL/I: POINTER

This operand is valid only for American National Standard (ANS) COBOL and PL/I programs. If omitted, the default is FBIN.

DATA2TP=
is similar to DATA1TP except that it is used for the second data field of the trace table entry.

TRACE TABLE

The CICS/VS trace table consists of a trace header and a variable number of fixed-length entries used to trace the logical flow of transaction activity through the system. Following generation, the trace feature may be invoked during system initialization by specifying the number of trace table entries to be other than zero. Optionally, the trace feature can be initially disabled during system initialization by the numbers of trace table entries (other than zero) that are to be off. These numbers are specified in the same initialization parameter. The master terminal facility can be used to turn specified traces on during CICS/VS execution. If a nonzero number of entries in the trace table is specified, the address of the trace header is placed at CSATRTBA.

Each entry in the trace table is 16 bytes in length and aligned on a double-doubleword boundary. The table is used in a wraparound manner so that when the last entry is used, the next entry is placed at the beginning of the table. The first three words of the table contain HEADER AT and the fourth word contains the address of the trace table header. The header format is:

Bytes	Contents
0-3	Address of the last-used entry
4-7	Address of the beginning of the table
8-11	Address of the end of the table
12-15	Reserved

The format of each succeeding entry in the trace table is:

Bytes	Contents
0	Trace identification of entry.
1-3	If byte 0 contains other than one of the values from X'F0' through X'FB', these bytes contain the contents of register 14 at entry to the trace control program. If byte 0 contains a value from X'F0' through X'FC', these bytes contain the contents of register 14 at entry to the CICS/VS management or service program involved.
4-5	If byte 0 contains one of the values for X'F0' through X'FC' (that is, if the trace identification is one of these values), this field generally contains the type of request code as it relates to the CICS/VS management or service program involved.

Program	Trace Identification
Task control	X'F0'
Storage Control	X'F1', X'C8' X'C9', X'CA'
Program control	X'F2'
Interval control	X'F3'
Dump control	X'F4'
File control	X'F5'
Transient data control	X'F6'
Temporary storage control	X'F7'

CICS/VS-DL/I interface	X'F8'
Journal control	X'F9'
Basic mapping support	X'CD',X'FA'
	X'CF'
Built-in functions	X'FB'
Terminal control	X'FC'
Sync point	X'D8'

For user entries, bytes 4 and 5 are unused.

6-7 Transaction identification as found in TCAKCTTA, a field in the system control section of the TCA. This identification is either a user task sequence number from 1 to 999, assigned by CICS/VS and stored in packed decimal form in the rightmost (low-order) bytes, or the system task identification (KC for task control, TC for terminal control, or JC for jorunal control) stored in the leftmost bytes.

8-11 Data field 1.

12-15 Data field 2.

The contents of bytes 0, 8 to 11, and 12 through 15 of application program-requested entries are determined by DFHTR TYPE=ENTRY macro instructions. The contents of trace table entries for the CICS/VS programs listed above are described in detail in Figures 8-1 through 8-13, which follow.

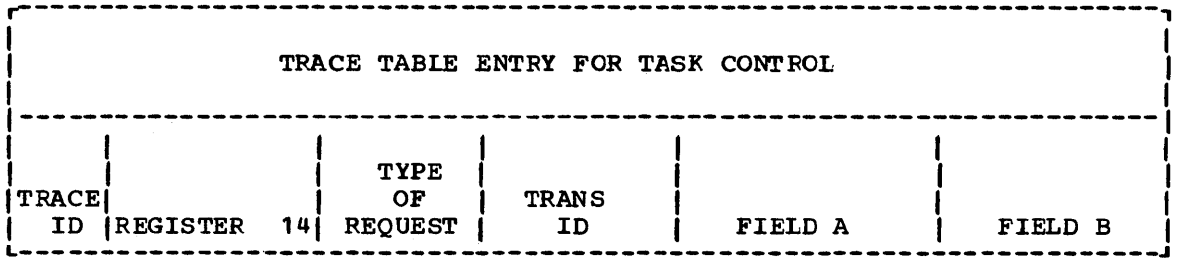
If consecutive, duplicate entries for the trace table are generated, the first entry has the form of a standard entry. Rather than creating multiple similar entries, however, a special trace control entry with trace identification X'FD' is created, and a count of the number of times the previous entry is repeated is stored therein (see Figure 8-14). Trace control entries with trace identification X'FE' or X'FF' indicate the turning on or turning off of the trace facility, respectively.

Some of the functions required for recovery/restart as available under CICS/OS/VS are performed by the sync point program. The trace table entry for this program is described in Figure 8-15.

An entry with a trace identification in the range from X'E6' through X'EF' is a special Field Engineering (FE) entry. The trace identification X'E6' is included in all entries produced by the terminal abnormal condition program. The contents and format of these entries are described in Figure 8-16.

If the entry is written to the auxiliary-trace data set, a 4-byte prefix is appended to the entry. This prefix contains the time that the entry was written to that data set. Auxiliary trace writes the time, to the data set, in units of 128 micro seconds. The trace utility program converts this time to hours: minutes: seconds, when formatting the auxiliary trace output.

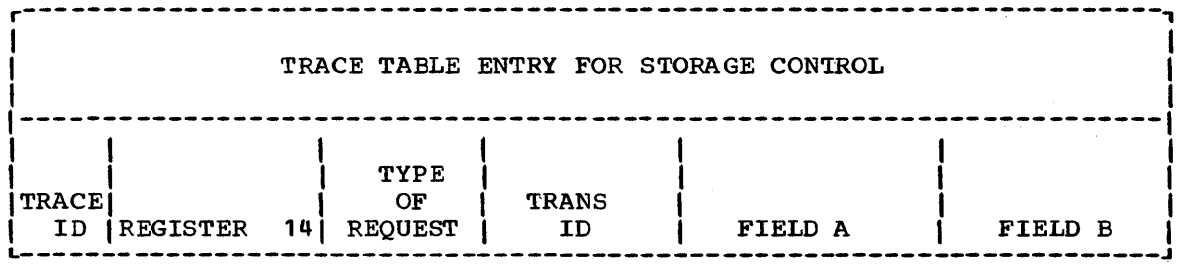
Note: The contents of any fields characterized as "Not used" in the descriptions that follow should be ignored during the analysis of a trace table entry.



X'C8', X'C9', X'CA': See Storage Control
X'CF': See Basic Mapping Support
X'D8': See Sync Point Program
X'E6': See Field Engineering

	Request Code (second byte not used)			
X'F0'	X'01'	(ENQ)	Queue name address TCATCQA	Not used
	X'02'	(DEQ)	Queue name address TCATCQA	Not used
	X'03'	(DEQALL)	Not used	Not used
	X'04'	(SUSPEND)	Not used	Not used
	X'08'	(RESUME)	TCA address of resumed transaction	Not used
	X'10'	(ATTACH)	Facility control address	Transaction ID TCAKCTI
	X'11'	(Conditional ATTACH)	Facility control address	Transaction ID TCAKCTI
	X'12'	(SCHEDULE)	Terminal ID or AID address TCAKCTA	Transaction ID TCAKCTI
	X'14'	(AVAIL)	Facility control address	Not used
	X'20'	(CHAP)	New priority TCATCDP	Not used
	X'40'	(WAIT)	Dispatch control indicator TCATCDC	Event control address TCATCEA
	X'80'	(DETACH)	Not used	Not used

Figure 8-1. Trace Table Entry for Task Control



Request Code
(second byte not used)

	Bit	Condition	Byte	
X'F1'	0	1=GETMAIN	If GETMAIN - 0 Not used	0-3 Facility address
			1 Initialization byte for GETMAIN	
			2-3 Requested number ot bytes	
	1	1=FREEMAIN if bit 0=0	If FREEMAIN - 0 Not used	0-3 Facility address
		1=Initialize storage if bit 0=1	1-3 Address of area to be freed	
	2	1=Release all terminal strg if bit 0=0 and bit 1=1	1-3 Not used if RELEASE=ALL	
		1=Conditional GETMAIN if bit 0=1		
		1=Cushion change if bit 0=0 and bit 1=0		
	3-7	Storage class		
		00000 1WD		
		00001 DCA		
		00010 QEA		
		00011 TQA		

Figure 8-2 (Part 1 of 2). Trace Table Entry for Storage Control

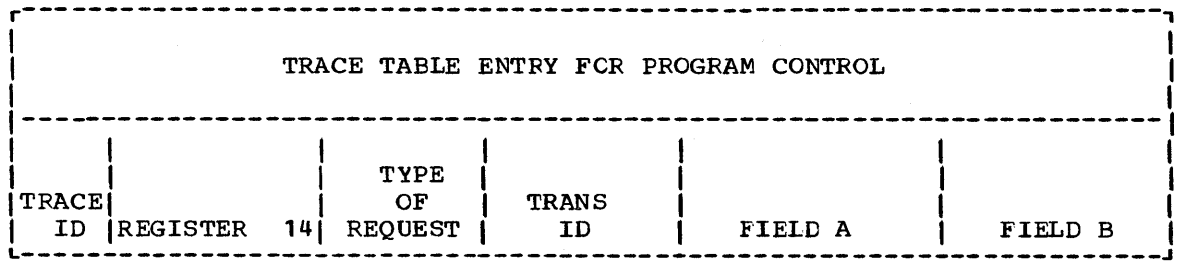
3-7 Storage class (continued)

00100	LINE
00101	TERMINAL
00110	ICE
00111	AID
01000	PROGRAM
01001	RSA
01010	TCA
01011	LLA
01100	USER
01101	TRANSDATA
01110	TEMPSTRG
01111	FILE
10001	WRE
10010	BCA
10011	SHARED
10100	CONTROL
10110	TACLE
10111	ISMAIN
11000	TSTABLE
11001	MAP
11010	ALIGNED
11011	JCA
11101	DWE
11110	MAPCCPY

Note: Many of these types of storage are used only by CICS/VS. The storage content is significant to system programmers and others who maintain CICS/VS.

X'C8'	Not used	0-3	Address of main storage acquired	Storage accounting
X'C9'	Not used	0-3	Address of main storage released	Storage accounting
X'CA'	Not used	0-3	Address of active TCA when storage control recovery was entered	Address in storage control where storage violation was encountered

Figure 8-2 (Part 2 of 2). Trace Table Entry for Storage Control



	Request Code		
X'F2'	X'0100' (LINK)	Program name from TCAPCPI	
	X'0200' (XCTL)	Program name from TCAPCPI	
	X'0400' (LOAD)	Program name from TCAPCPI	
	X'0800' (DELETE)	Program name from TCAPCPI	
	X'1000' (RETURN)	Program name from TCAPCTA	
	X'2000' (SETXIT, CANCEL)	Not used	Not used
	X'2001' (SETXIT, PROGRAM)	Program name from TCAPCPI	
	X'2002' (SETXIT, ROUTINE)	Routine address from TCAPCERA	Not used
	X'2004' (BLDL)	Program name from TCAPCPI	
	X'2008' (RESETXIT)	Not used	Not used
	X'2400' (LOAD, LOADLST=NO)	Program name from TCAPCPI	
	X'4000' (ABEND without dump)	Not used	Not used
	X'4100' (ABEND, CANCEL=YES)	Not used	Not used
	X'6000' (ABEND with dump)	ABEND code from TCAPCAC	Not used
	X'6100' (ABEND with dump, CANCEL=YES)	ABEND code from TCAPCAC	Not used
	X'8100' (Cond LINK)	Program name from TCAPCPI	
	X'8200' (LOCATE)	Program name from TCAPCPI	
	X'8400' (Cond LOAD)	Program name from TCAPCPI	
	X'A400' (Cond LOAD, LOADLST=NO)	Program name from TCAPCPI	

Figure 8-3. Trace Table Entry for Program Control

TRACE TABLE ENTRY FOR INTERVAL CONTROL

TRACE ID	REGISTER 14	TYPE OF REQUEST	TRANS ID	FIELD A	FIELD B
X'F3'		X'1x' (GETIME)		Return time to user address TCAICDA	Not used
		where "x" consists of the low-order four bits: Bit Condition 4,5 Always zero 6 0=Refresh CSA time only 1=Return time to user 7 0=Binary format 1=Packed format			
		X'2x' (WAIT)		INTRVAL or TIME value (TCAICRT)	Not used
		X'3x' (POST)		INTRVAL or TIME value (TCAICRT)	Not used
		where "x" consists of the low-order four bits: Bit Condition 4 0=INTRVAL parameter provided 1=TIME parameter provided 5 0=No request ID provided 1=User-provided request ID 6,7 Always zero			
		X'4x' (INITIATE)		INTRVAL or TIME value (TCAICRT)	Transaction ID (TCAICTI)

Figure 8-4 (Part 1 of 2). Trace Table Entry for Interval Control

Request Code (second byte not used)		
X'5x' (PUT)	INTRVAL or TIME value (TCAICRT)	Transaction ID (TCAICTI)

where "x" consists of the low-order four bits:

Bit	Condition
4	0=INTRVAL parameter provided 1=TIME parameter provided
5	0=No request ID provided 1=User-provided request ID
6	Always zero
7	0=Nonterminal destination 1=Terminal destination

X'8x' (GET)	User-provided data address	Not used

where "x" consists of the low-order four bits:

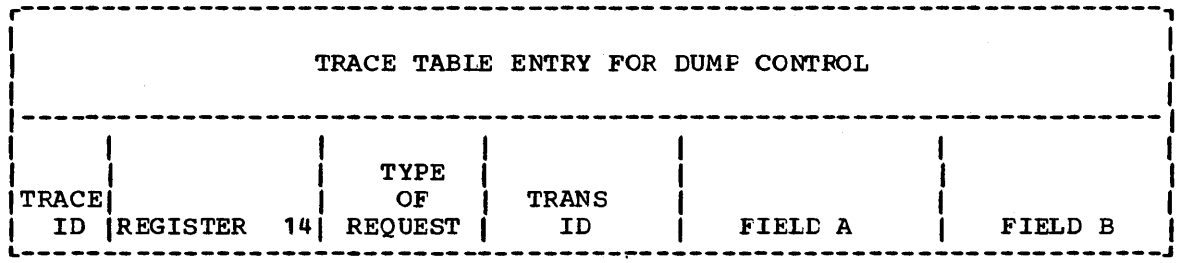
Bit	Condition
4,5	Always zero
6	0=User-provided data address 1=Return data address to user
7	Always zero

X'90' (RETRY)	Request ID	(TCAICQID)
X'A0' (RESET)	Not used	Not used
X'Fx' (CANCEL)	Request ID	(TCAICQID)

where "x" consists of the low-order four bits:

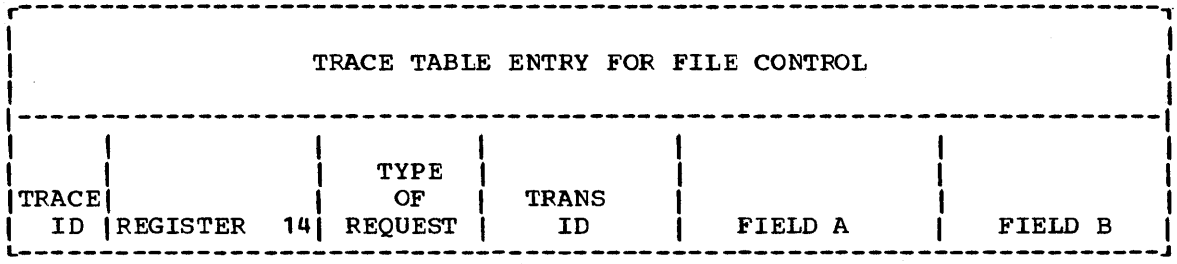
Bit	Condition
4	Always zero
5	0=No request ID provided 1=User-provided request ID
6,7	Always zero

Figure 8-4 (Part 2 of 2). Trace Table Entry for Interval Control



	Request Code			
X'F4'	X'FE00'	TRANSACTION	Not used	Dump code
	X'00FF'	CICS	Not used	
	X'FEFF'	COMPLETE	Not used	Dump code
		PARTIAL		
	X'0100'	SEGMENT	Not used	Dump code
	X'0400'	TRANSACTION	Not used	Dump code
	X'0800'	TERMINAL	Not used	Dump code
	X'20000'	PROGRAM	Not used	Dump code

Figure 8-5. Trace Table Entry for Dump Control



	Request Code		
X'F5'	X'80' GET may also include: X'08' Segmented X'04' Update X'02' Indirect access X'01' Deblock direct access X'80' Deblock by key (see note 1) X'40' Deblock by relative record (see note 1)	Data set name from TCAFCDI or FCT entry (for all except RELEASE, OPEN, CLOSE, and LOCATE).	
	X'40' PUT may also include: X'08' Segmented X'04' New record X'01' Delete		
	X'20' GETAREA may also include: X'08' Initialize storage X'04' Mass insert		
	X'10' RELEASE	Area address from TCAFCAA	Not used
	X'01' DELETE		
	X'00' System RELEASE	Area address from TCAFCAA	Not used
	X'A0' SETL may also include: X'08' Segmented		
	X'A4' RESETL may also include: X'08' Segmented		
	X'B0' GETNEXT may also include: X'08' Segmented		
	X'11' ESETL		
	X'C0' OPEN	Pointer to list	List

Figure 8-6 (Part 1 of 2). Trace Table Entry for File Control

Request Code			
X'E0'	CLOSE	Pointer to list	List
X'F0'	LOCATE	Pointer to list	List

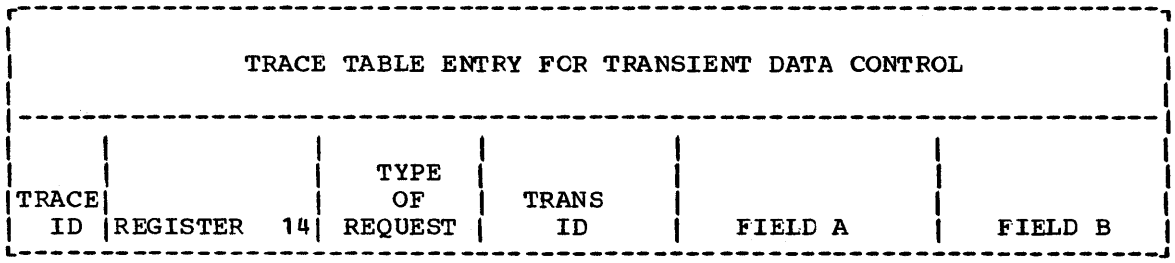
Notes:

1. Contents of second byte; applies only to deblock direct access data sets.

2. If the first byte of the request code is X'80', X'40', X'20', X'11', X'10', X'01', X'A0', X'A4', or X'B0', and a VSAM data set is being processed, the second byte of the request code contains one of the following:

X'80'	Argument is RBA
X'40'	Argument is generic key
X'20'	Search greater than or equal to
X'10'	Locate mode

Figure 8-6 (Part 2 of 2). Trace Table Entry for File Control



	Request Code (second byte not used)		
X'F6'	X'04' (PURGE)	Not used	Destination ID from TCATDDI
	X'10' (LOCATE)	Not used	Not used
	X'20' (FEOV)	Not used	Destination ID from TCATDDI
	X'40' (PUT)	Data address from TCATDDA	Destination ID from TCATDDI
	X'48' (PUT)	Issued by the asynchronous transaction control program (DFHATP); see PUT above	
	X'80' (GET)	Not used	Destination ID from TCATDDI
	X'88' (GET)	Issued by the asynchronous transaction control program (DFHATP); see GET above	

Figure 8-7. Trace Table Entry for Transient Data Control

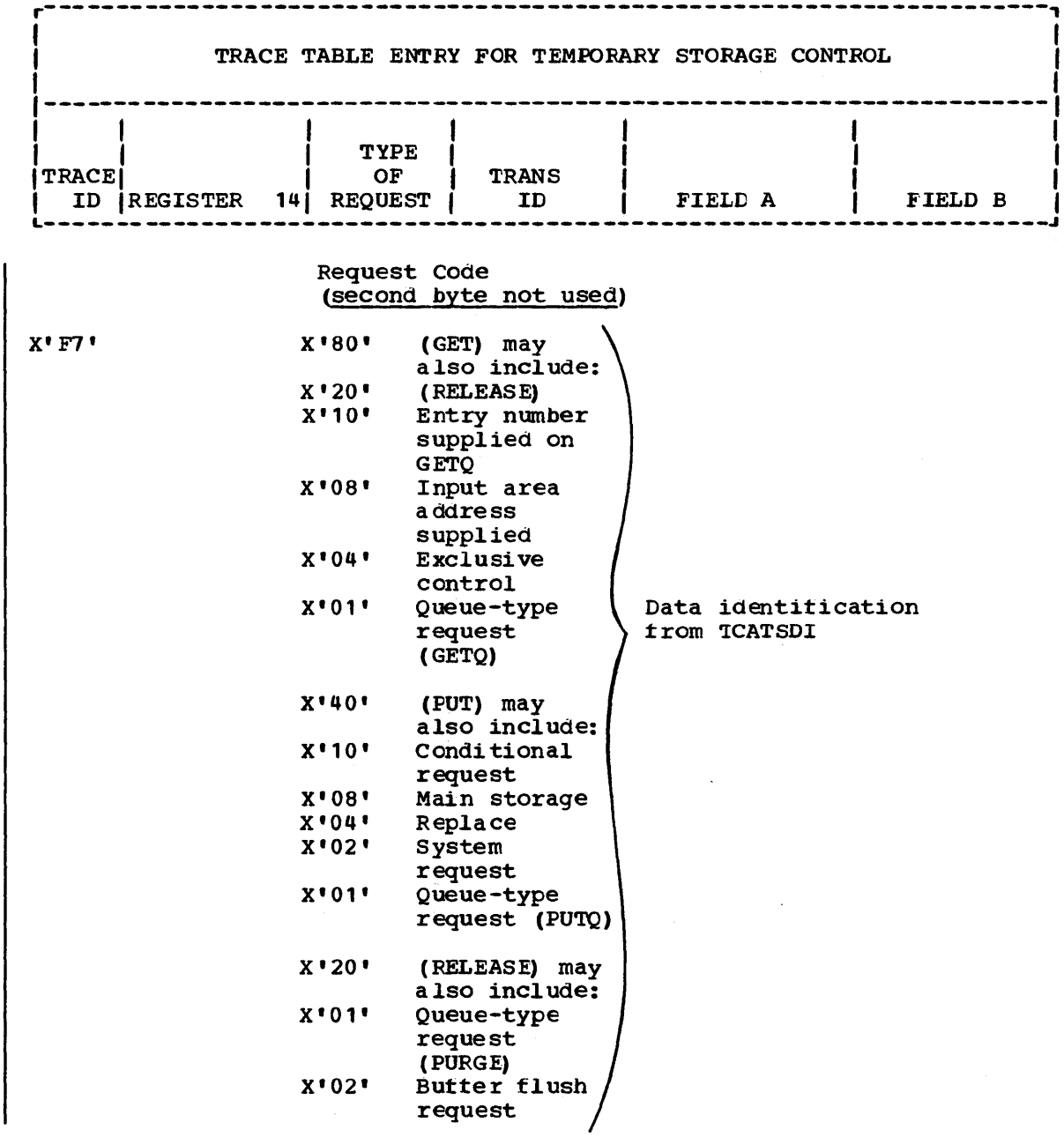
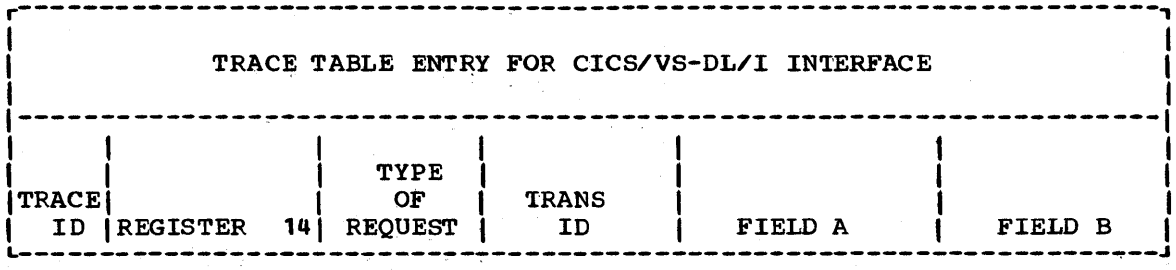
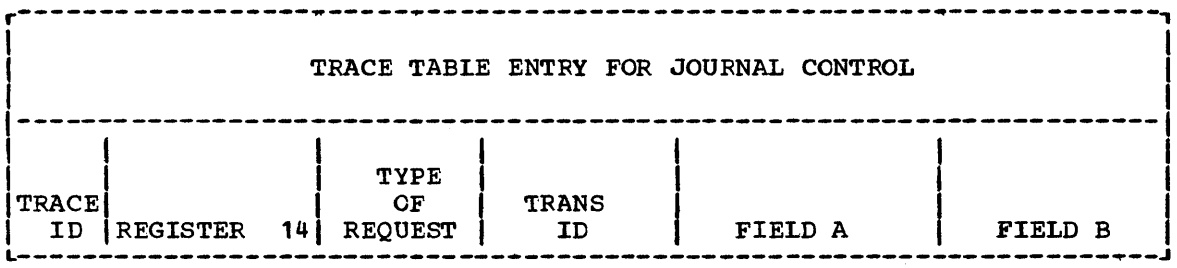


Figure 8-8. Trace Table Entry for Temporary Storage Control



Request Code			
X'F8'	Not used	DL/I Function code	PCB address

Figure 8-9. Trace Table Entry for CICS/VS-DL/I Interface



Request Code (not used as request code)	Bytes	Contents
	FIELD A:	
X'F9'	0	Journal file ID
	1	Not used
	2-3	Type of request

Note: Byte 3 is the major control; byte 2 further identifies the condition.

Byte 3	Byte 2	
X'01'		WRITE
	X'01'	COND=YES
	X'02'	STARTIO=YES
	X'04'	User prefix specified
	X'10'	USING clause passes data address (COBOL)
	X'80'	CICS/VS request
X'02'		WAIT
	X'02'	STARTIO=YES
X'03'		PUT
	Same as WRITE	
X'04'		OPEN
	X'01'	Output
	X'02'	Input
	X'04'	VOL=FIRST (output) or VOL=PREVIOUS (input)
	X'08'	VOL=NEXT
	X'10'	VOL=CURRENT or SIVOL=YES (VOL=NEXT)
X'08'		CLOSE
	X'01'	LEAVE=YES
X'10'		NOTE
X'20'		POINT
X'40'		GETF
X'80'		GETB

FIELD B:	
4-7	Address of JCA

Figure 8-10. Trace Table Entry for Journal Control

TRACE TABLE ENTRY FOR BASIC MAPPING SUPPORT

TRACE ID	REGISTER 14	TYPE OF REQUEST	TRANS ID	FIELD A	FIELD B
X'CD'		Not used		Bytes 0-3 of Temporary Storage data identification (TCATSDI)	Bytes 4-7 of Temporary Storage data identification (TCATSDI)

Note: The X'CD' trace occur when BMS receives an IDERROR from Temporary Storage. This trace can be used by Field Engineering.

Figure 8-11 (Part 1 of 3). Trace Table Entry for Basic Mapping Support

X'FA'	Not used	Request Code

FIELD A:

Byte 0 (from TCAMSTR1)	Meaning
X'80'	TYPE=ROUTE
X'40'	ERRTERM=ORIG
X'20'	ERRTERM=termid
X'10'	INTERVAL=numeric
X'08'	TIME=numeric
X'04'	LIST=ALL
X'02'	LIST=symbolic address
X'01'	OPCLASS=operator class

Byte 1 (from TCAMSTR2)	Meaning
X'80'	TITLE=symbolic address
X'01'	TYPE=PURGE

Byte 2 (from TCAMSTR3)	Meaning
X'20'	TYPE=TEXT
X'10'	CURSOR=number
X'08'	CTRL=any 3270 WCC
X'04'	MAP=map name
X'02'	MSETADR=symbolic address
X'01'	MAPSET=map set name

Byte 3 (from TCAMSTR4)	Meaning
X'C0'	DATA=YES
X'40'	DATA=NO
X'20'	TYPE=SAVE
X'10'	MAPADR=symbolic address
X'08'	TYPE=WAIT
X'04'	TYPE=MAP
X'02'	TYPE=ERASE
X'01'	TYPE=IN

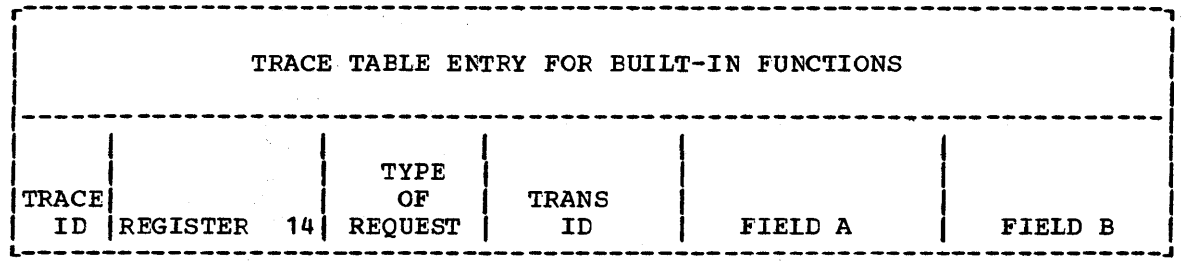
FIELD B:

Byte 4 (from TCAMSTR5)	Meaning
X'80'	TYPE=PAGEBLD
X'40'	OFLOW=symbolic address
X'04'	TYPE=OUT
X'02'	TYPE=STORE
X'01'	TYPE=RETURN

Figure 8-11 (Part 2 of 3). Trace Table Entry for Basic Mapping Support

		Request Code Byte 5 Meaning (from TCAMSTR6) X'80' TYPE=PAGEOUT X'40' CTRL=AUTOPAGE X'20' CTRL=PAGE X'10' CTRL=RETAIN X'08' CTRL=RELEASE X'04' WRBRK=CURRENT X'02' WRBRK=ALL X'01' EODPURG=OPER Byte 6 Meaning (from TCAMSTR7) X'80' TYPE=TEXTBLD X'40' HEADER=symbolic address X'20' TRAILER=symbolic address X'10' JUSTIFY specified X'01' TYPE=NOEDIT Byte 7 Not used
X'CF'	Not used	FIELD A: Byte 0 Response code from TCAMSRC1: X'80' Route failed - no resolutions X'40' Route worked - some resclutions X'20' Invalid error terminal X'08' Map too large X'04' I/O area cannot be mapped X'02' Page returned X'01' Invalid request X'00' Normal response Byte 1 Response code from TCAMSRC2: X'80' Temporary storage I/O error Byte 2 Response code from TCAMSRC3: X'01' PAGEBLD overflow Byte 3 Terminal code from TCAMSRI1 when TCAMSRC1 contains X'08' FIELD B: When TCAMSRC3 contains X'01', the first two bytes of this field contain the current page number (from TCAMSPGN) and the last two bytes contain the overflow control number (from TCAMSOCN).

Figure 8-11 (Part 3 of 3). Trace Table Entry for Basic Mapping Support

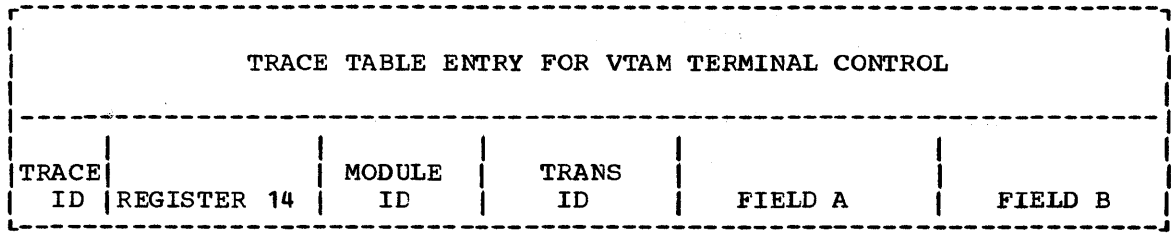


	Request Code (second byte not used)		Byte	Contents
				FIELD A includes bytes 0-3; FIELD B includes bytes 4-7.
X'FB'	X'01'	BITEST	0	Not used Byte address
			1	Function type X'80' BITSETON X'40' BITSETOFF X'20' BITFLIP X'10' BITEST
			2	Bit pattern
			3	Not used
	X'02'	DEEDIT	0-1	Field length Field address
			2-3	Not used
	X'03'	INFORMAT	0	Not used List address
			1	Name list indicator X'00' No list
			2-3	TIOA size
	X'04'	PHONETIC		Bytes 0-7 of the name
	X'05'	CHECK	0-1	Field length Field address
			2-3	Not used
	X'06'	TSEARCH	0	Not used Arg. address
			1	Function code X'00' EBCDIC X'01' Target address X'10' Descending order X'20' Ascending order X'40' Range
			2-3	Number of entries in arg. table

Figure 8-12 (Part 1 of 2). Trace Table Entry for Built-In Functions

Request Code (second byte not used)		Byte	Contents	
X'07'	WTRETST	0	Not used	Key address
		1	SETL indicator	
		2-3	Max. number of records	
X'08'	WTRTPARM	0-1	Not used	VSWA address
		2-3	FIELD1 - second operand	
X'09'	WTRETGET	Same as WTRETST		VSWA address
X'0A'	WTRETREL	Same as WTRETST		VSWA address

Figure 8-12 (Part 2 of 2). Trace Table Entry for Built-In Functions



	Module ID		Field A	Field B
X'FC'	X'00'	Exit trace entry	Contents are residual	Format B1
	X'01'	Send DFSYN	Format A1	Format B1
	X'02'	Send DFASY	Format A1	Format B1
	X'03'	Send response	Format A1	Format B1
	X'04'	Receive specitic	Format A1	Format B1
	X'05'	SIMLOGON	Format A1	Format B1
	X'06'	OPNDST	Format A1	Format B1
	X'07'	SESSIONC	Format A1	Format B1
	X'08'	CLSDST	Format A1	Format B1
	X'09'	RESETSR	Format A1	Format B1
	X'0B'	Appl request	Format A2	Format B2 it byte 0 ot tield A is X'00', otherwise not used.
	X'0C'	Receive any completion	Format A1	Format B1
	X'0D'	CTYPE and request	Format A1	Format B1
	X'22'	Auto task initiation	Format A1	Format B1
	X'23'	Start-up-task	Format A1	Format B1
	X'24'	DWE process	Format A1	Format B1
	X'25'	Resync	Format A1	Format B1
	X'26'	Resp logger	Format A1	Format B1
	X'27'	NACP	Format A3	Format B1
	X'29'	NACP	Format A4	Format B1
	X'31'	Auto queueing	Format A1	Format B1
	X'32'	Detach	Format A1	Format B1
	X'34'	Locate	Format A1	Format B1
	X'40'	Attach	Format A1	Format B1
	X'80'	GETMAIN	Format A1	Format B1
	X'C0'	FREEMAIN	Format A1	Format B1

FORMAT A1

Byte 0	Meaning
X'01'	Task created by avail (ATI)
X'02'	Input journal required tlag
X'08'	Resynch/recovery in progress
X'10'	NACP in progress
X'20'	ATI bid in progress
X'40'	Data in progress
X'80'	Command in progress

Figure 8-13 (Part 1 of 7). Trace Table Entry for VTAM Terminal Control

Byte 1	Meaning
X'01'	Resynchronization required
X'02'	Previous session ABEND
X'04'	Emergency restart
X'08'	Bracket protocol required
X'10'	Overlength data
X'20'	Mode (CS=X'20', CA=Not X'20')
X'40'	CICS quiesced by node
X'80'	Node quiesced by CICS

Byte 2	Meaning
X'01'	Definite resp sent in progress
X'02'	Bid to be retried indicator
X'04'	Log 1st input after sync
X'08'	Not used
X'10'	Awaiting positive response
X'20'	Deferred write pending
X'40'	Pending FME response
X'80'	Pending RRN response

Byte 3	Meaning
X'08'	End bracket sent
X'10'	Begin bracket sent
X'20'	RTR pending state
X'40'	Begin bracket pending state
X'80'	In bracket state

FORMAT A2

Byte 0	Meaning
X'00'	I/O request type

Figure 8-13 (Part 2 of 7). Trace Table Entry for VTAM Terminal Control

Byte 1	Meaning
X'01'	Write request
X'02'	Converse request
X'04'	Synchronization request
X'08'	Disconnect request
X'10'	Read request
X'20'	Line addressing request
X'40'	Save terminal storage request
X'80'	Erase request

Byte 2 Not used

Byte 3	Meaning
X'01'	Pseudo binary mode
X'02'	Notranslate request
X'02'	Transparent mode
X'04'	Print request
X'08'	Copy request
X'10'	Read lock request
X'20'	Write lock request
X'40'	Erase all unprotected
X'80'	Read butter request

Byte 0	Meaning
X'01'	Locate request

Byte 1	Meaning
X'00'	Address request
X'01'	ID request
X'02'	Next request

Bytes 2-3 Not used

Byte 0	Meaning
X'02'	Auto task initiation request

Bytes 1-3 Not used

Byte 0	Meaning
X'04'	Status request

Byte 1	Meaning
X'01'	Acquire
X'02'	Resynchronization override
X'04'	Release

Figure 8-13 (Part 3 of 7). Trace Table Entry for VTAM Terminal Control

Byte 2	Meaning
X'01'	Out of service
X'02'	In service
X'04'	Transaction
X'08'	Auto initiate
X'08'	Transceive
X'10'	No poll
X'10'	Receive
X'20'	Input
X'40'	Auto page
X'80'	Page

Byte 3 Not used

Byte 0	Meaning
X'08'	LDC request

Byte 1 Contains the logical device code

Bytes 2-3 Not used

Byte 0	Meaning
X'10'	Detach request

Bytes 1-3 Not used

Byte 0	Meaning
X'20'	Sync-point request

Bytes 1-3 Not used

Figure 8-13 (Part 4 of 7). Trace Table Entry for VTAM Terminal Control

FORMAT A3

Byte 0	Meaning
X'10'	NODE not activated
X'11'	Session bind failure
X'13'	VTAM halting
X'14'	Logic error detected
X'15'	Permanent channel error or NCP shutdown
X'19'	Terminate self from NODE
X'1A'	Apparent VTAM error
X'1D'	VTAM inactive to application or TCE ABEND
X'20'	VTAM inactive
X'44'	Exception response received
X'59'	Record outstanding at shutdown
X'60'	Unsupported command detected
X'65'	Invalid response requested
X'73'	Read only terminal
X'74'	Unsupported command
X'75'	Unsupported command
X'78'	Response request error
X'80'	Temp VTAM storage problem
X'81'	Exception response received
X'82'	Unknown command received
X'83'	ATI no longer requested
X'84'	Invalid normal response to bid
X'88'	Invalid CID detected
X'89'	Unknown symbolic name
X'90'	ZCP logic error detected
X'91'	Invalid RTYPE specified
X'92'	Send DFSYN req incomplete
X'93'	CA mode and task attached
X'94'	Input status error
X'95'	TIOA length incorrect
X'96'	RPL missing (receive specific)
X'97'	TIOA missing (receive specific)
X'98'	No task on terminal to resume
X'99'	No TIOA available for send
X'A1'	Invalid read request
X'A3'	Bracket state error
X'A5'	Message exceeds input maximum
X'A7'	NODE bracket protocol error
X'A8'	FMH length > data length
X'A9'	Receive specific in receive any failed
X'B1'	RPL is active
X'B2'	Invalid command setting
X'B3'	No RPL exists for operation
X'B5'	Unknown command
X'BA'	Exception response
X'BB'	Unknown command in RPL
X'C1'	Unknown error code from VTAM
X'C4'	Dummy TCTTE identifier
X'C5'	NCP restarted
X'CB'	Terminal CLSDST'D
X'CC'	Clear was issued
X'CD'	Exception in chain

Figure 8-13 (Part 5 of 7). Trace Table Entry for VTAM Terminal Control

Byte 1	Meaning
X'80'	VTAM recovered NODE
X'40'	NODE unrecoverable
X'20'	NODE recovery in progress
X'08'	Request recovery record
X'04'	Unsupported command received
X'02'	CICS released by node
X'01'	CICS quiesced by node

Byte 2	Meaning
X'80'	Catastrophic error
X'40'	Exception req received
X'20'	Request shutdown record - task active
X'08'	Negative response record to a definite resp send
X'04'	Exception response req

Byte 3	Meaning
X'01'	Release request received
X'02'	Shutdown sent by CICS
X'04'	Committed log pending
X'20'	Start data traffic sent
X'40'	Session bind
X'80'	Node is logged on

FORMAT A4

Byte 0	Meaning
X'20'	Print TCTTE
X'40'	Print RPL
X'80'	Print action flags

Byte 1	Meaning
X'20'	ABEND task
X'40'	ABORT VTAM receive
X'80'	ABORT VTAM send

Byte 2	Meaning
X'01'	Terminate session
X'02'	Keep node out of service

Byte 3	Meaning
X'01'	Definite resp send
X'02'	Bid to be retried
X'04'	Log first input
X'08'	0=positive 1=negative
X'10'	Awaiting positive response
X'20'	Deferred write pending
X'40'	Pending FME response
X'80'	Pending RRN response

Figure 8-13 (Part 6 of 7). Trace Table Entry for VTAM Terminal Control

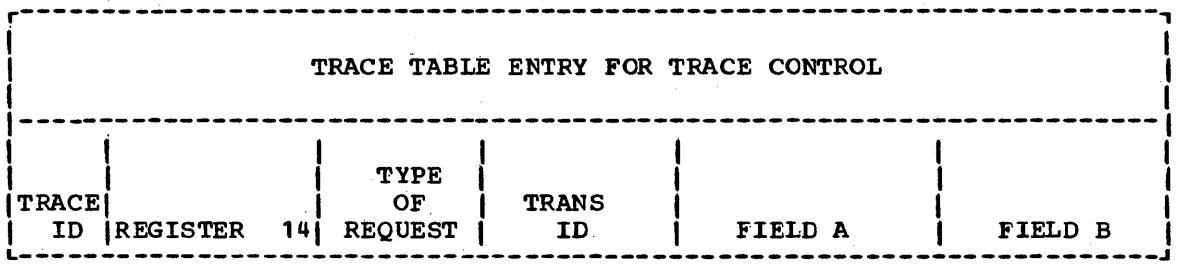
FORMAT B1

Byte 0	Meaning
X'00'	No exit since last trace entry
X'E2'	DFASY exit
X'E3'	SESSIONC exit
X'E4'	SESSIONC input exit
X'E5'	LOSTERM exit
X'E6'	TPEND exit
X'E7'	Release request exit
X'E8'	LERAD exit
X'E9'	SYNAD exit
X'F1'	Send DFSYN exit
X'F2'	Send DFASY exit
X'F3'	Response exit
X'F4'	Receive specific exit
X'F5'	LOGON exit
X'F6'	OPNDST exit
X'F8'	CLSDST exit
Bytes 1-3	Contain the address of the TCTTE

FORMAT B2

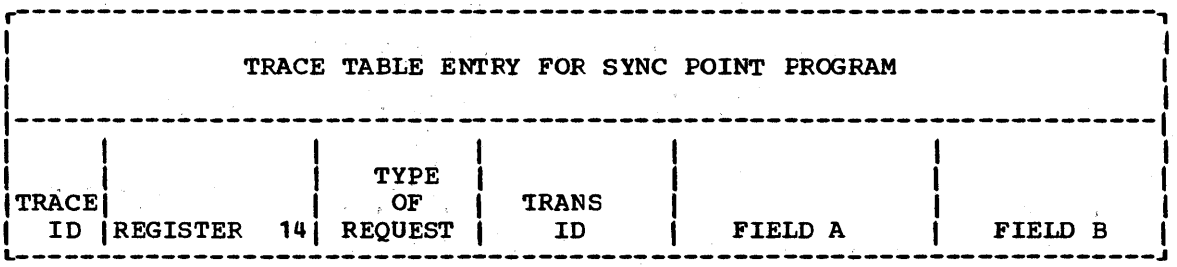
Byte 0	Meaning
X'01'	Wait request with operation
X'02'	Override sync operation
X'04'	Override async operation
X'08'	Last write from task
X'10'	FMH provided with data
X'20'	LDC mnemonic present
Bytes 1-3	Contain the address of the TCTTE

Figure 8-13 (Part 7 of 7). Trace Table Entry for VTAM Terminal Control



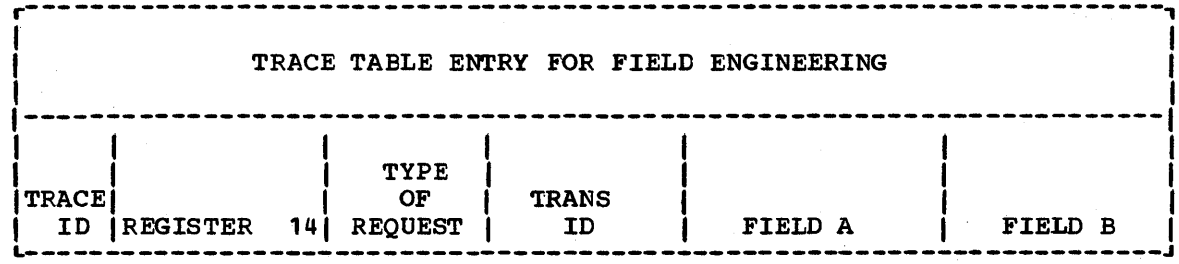
Request Code					
X'FD'	Not used	Number of repeated entries (packed decimal) in trace table			
		Byte	Contents	Byte	Contents
X'FE' (Trace turn on)		0	CSATRMF1	0	TCATRTR
		1	CSATRMF2	1	Reserved
X'FF' (Trace turn off)		2	CSATRMF3	2	Reserved
		3	TCATRMF	3	Reserved

Figure 8-14. Trace Table Entry for Trace Control



Request Code			
X'D8'	Not used	Bytes (from TCASPTR)	Bytes (from TCADWLBA)
		0 Type of request X'01' USER X'02' SINGLE	0-3 Pointer to first Deferred work element (DWE)
		1-3 Not used	

Figure 8-15. Trace Table Entry for Sync Point Program (CICS/OS/VS Only)



	Request Code	Byte	Contents
X'E6'	X'0100' (from TCATRID1)	0,1	Op code from field TCTLETOP of terminal control table line entry (TCTLE) for communication line
		2	Teleprocessing code from TCTLETPO
		3	Action flags from TCTLEECB+1 (set by abnormal termination condition program)
		4-7	Terminal identification

Note: This trace occurs in the terminal abnormal condition program (DFHTACP) just prior to transferring control to the user's terminal error program (DFHTEP).

	X'0200' (from TCATRID1)	0	Completion code from TCTLEECB
		1	BTAM return code from TCTLEECB+3
		2	Error flags from TCTLEPLF
		3	Action flags from TCTLEECB+1
		4	Command code from TCTLECC
		5	Status byte from TCTLESF
		6	First sense byte from TCTLESB
		7	Second sense byte from TCTLESB+1

Note: This trace occurs in DFHTACP just after returning from DFHTEP.

X'E7' through X'EF'; Reserved for field engineering.

Figure 8-16. Trace Table Entry for Field Engineering (FE) Type of Entry

TRACE TABLE ENTRY FOR TASK CONTROL
(During Auxiliary Tracing Only)

TRACE ID	REGISTER 14	TYPE OF REQUEST	TRANS ID	FIELD A	FIELD B
X'D0'		(second byte not used)			
		X'05' Task dispatched		Not used	Not used
		X'06' Task created		Facility ID	Transaction ID
		X'07' Task terminated		Not used	Not used

This trace entry is only created if auxiliary trace is active.

Figure 8-17. Trace Table Entry for Task Control (During Auxiliary Tracing Only)

DUMP SERVICES

Dump management provides the capability, through dump control, to dump specified areas of main storage onto a sequential data set, either tape or disk. This data set contains only the information pertinent to the user's transaction or application program, and is subsequently formatted and printed offline (or while the dump data set is closed) using a CICS/VS dump utility program (DFHDUP).

Requests for dump services are communicated to dump control through CICS/VS macro instructions. Dump control executes at the priority of the requesting program, under control of the TCA of the requesting program. saving and restoring registers from this TCA. After a requested dump service has been provided, control is returned to the next executable instruction in the requesting program.

Dump control operates as a serially reusable program resource. Only one service request is processed at a time. If additional requests for dump services are made while a dump is in progress, the tasks associated with those service requests are delayed (suspended) and placed in a "hold" status until the dump is completed. Remaining dump requests are serviced on a first-in first-out (FIFO) basis.

The dump management macro instruction (DFHDC) is used to request any of the following services:

- Dump main storage areas related to a transaction and its associated task (or any other main storage areas)
- Dump the following CICS/VS control tables: program control table (PCT), program processing table (PPT), system initialization table (SIT), terminal control table (TCT), file control table (FCT), and destination control table (DCT)
- Dump transaction-oriented storage areas and CICS/VS control tables
- Dump selected main storage areas related to the requesting task

Notes:

1. To ensure a dump of the TIOA following a terminal control write that precedes a DFHDC macro instruction, the application programmer must issue a SAVE and WAIT with the DFHTC TYPE=WRITE macro instruction.
2. If the purpose of the DFHDC macro instruction is to inspect the contents of the common control area of the application program communication section of the TCA used for a CICS/VS service request, the application program must acquire 24 words of storage (or use a portion of the TWA) prior to issuing the DFHDC macro instruction in which to save this information. Since the communication area in the TCA of the requesting task is used by dump control, the information in that area prior to the dump is overlaid during the dump.
3. CICS/VS control tables will be dumped only if the CICSDMP=YES operand is specified in the DFHSG PROGRAM=DCP macro instruction at system generation. (See the CICS/VS System Programmer's Reference Manual.)
4. Every dump request will include the TCA, CSA, and TRT.

DUMP TRANSACTION STORAGE (TRANSACTION)

The application programmer can request a dump of all main storage areas related to a transaction and its associated task by issuing the DFHDC TYPE=TRANSACTION macro instruction. This dump is normally used during the testing and debugging of user-written application programs. (CICS/VS automatically provides this service if the related task is abnormally terminated.)
The format of this macro instruction is:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHDC │ TYPE=TRANSACTION                                     │
│      │       │ ,DMPCODE={value}                                     │
│      │       │          {YES  }                                     │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=TRANSACTION
indicates that the following storage areas are to be dumped:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)
2. Common system area (CSA), including the user's portion of the CSA (CWA)
3. Trace table
4. Contents of general-purpose registers upon entry to dump control from requesting task
5. Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task
6. All transaction storage areas chained off the TCA storage accounting field
7. All program storage areas containing user-written application program(s) active on behalf of the requesting task
8. Register save areas (RSAs) indicated by the RSA chain off the TCA
9. All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task (if any)

Whenever the TCTTE is dumped (see 5 above), the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped. The latter are used by basic mapping support.

DMPCODE=
is a four-character dump code to be printed out with the requested dump to identify it; this code should be unique so that it is informative concerning the condition that caused the dump.

value
is a combination of four alphabetic, numeric, and/or special characters to be printed as the dump code.

YES
indicates that the dump code has been placed in TCADCDC.

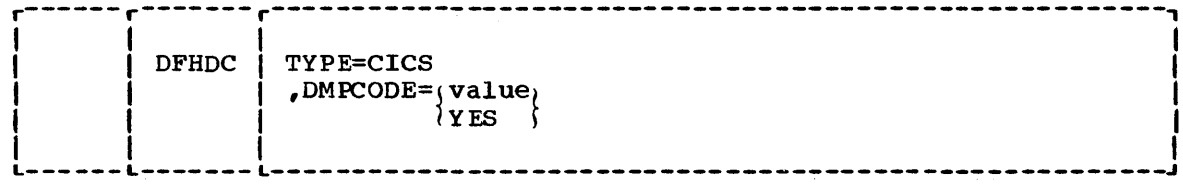
Example

The following example illustrates the coding required to request a dump of transaction storage:

```
DFHDC TYPE=TRANSACTION,     REQUEST TRANSACTION STORAGE DUMP      *
      DMPCODE=D010          USER-SPECIFIED DUMP CODE
```

DUMP CICS/VS STORAGE (CICS)

The application programmer can request a dump of PCT, PPT, TCT, FCT, and DCT by issuing the DFHDC TYPE=CICS macro instruction. This facility is available if the CICSDMP=YES operand is specified in the DFHSG PROGRAM=DCP macro instruction at system generation (see the CICS/VS System Programmer's Reference Manual). This dump is typically the first dump taken in a testing situation in which the base of the test must be established; subsequent dumps are usually of the TRANSACTION type.
The format of this macro instruction is:



where:

TYPE=CICS
indicates that PCT, PPT, SIT, TCT, FCT, and DCT are to be dumped.

DMPCODE=
is a four-character dump code to be printed out with the requested dump to identify it; this code should be unique so that it is informative concerning the condition that caused the dump.

value
is a combination of four alphabetic, numeric, and/or special characters to be printed as the dump code.

YES
indicates that the dump code has been placed in TCADCDC.

Example

The following example illustrates the coding required to request a dump of PCT, PPT, SIT, TCT, FCT, and DCT:

```
DFHDC TYPE=CICS,            REQUEST CICS/VS STORAGE DUMP      *
      DMPCODE=D020          USER-SPECIFIED DUMP CODE
```

DUMP TRANSACTION STORAGE AND CICS/VS STORAGE (COMPLETE)

The application programmer can request a dump of both transaction/task-related storage and PCT, PPT, SIT, TCT, FCT, and DCT

by issuing the DFHDC TYPE=COMPLETE macro instruction. The PCT, PPT, SIT, TCT, FCT, and DCT will be dumped if CICSDMP=YES was specified in the DFHSG PROGRAM=DCP macro instruction at system generation (see the CICS/VS System Programmer's Reference Manual).
The format of this macro instruction is:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │ DFHDC │ TYPE=COMPLETE                                        │
│      │       │ ,DMPCODE={value}                                     │
│      │       │          {YES  }                                     │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=COMPLETE
indicates that transaction/task-related storage and PCT, PPT, SIT, TCT, FCT, and DCT are to be dumped.

DMPCODE=
is a four-character dump code to be printed out with the requested dump to identity it; this code should be unique so that it is informative concerning the condition that caused the dump.

value
is a combination of four alphabetic, numeric, and/or special characters to be printed as the dump code.

YES
indicates that the dump code has been placed in TCADCDC.

To request a complete dump is sometimes appropriate during execution of a task, but this macro instruction should not be used excessively. CICS/VS control tables are primarily static areas; therefore, requesting one CICS dump and a number of TRANSACTION dumps is generally more efficient than requesting a comparable number of COMPLETE dumps.

Example

The following example illustrates the coding required to request a dump of both transaction storage and PCT, PPT, SIT, TCT, FCT, and DCT:

```
      DFHDC TYPE=COMPLETE,          REQUEST COMPLETE STORAGE DUMP      *
            DMPCODE=D030            USER-SPECIFIED DUMP CODE
```

DUMP PARTIAL STORAGE (PARTIAL)

The application programmer can request a dump of selected main storage areas related to the requesting task by issuing the DFHDC TYPE=PARTIAL macro instruction. This type of dump can be used during the testing and debugging of user-written application programs. It includes only the storage areas specified.

The format of this macro instruction is:

```
┌──────┬───────┬──────────────────────────────────────────────────────────┐
│      │       │                                                          │
│      │ DFHDC │ TYPE=PARTIAL                                             │
│      │       │ LIST=([TERMINAL][,PROGRAM][,TRANSACTION][,SEGMENT])      │
│      │       │ ,DMPCODE={value}                                         │
│      │       │          {YES  }                                         │
│      │       │                                                          │
└──────┴───────┴──────────────────────────────────────────────────────────┘
```

where:

TYPE=PARTIAL
indicates that a dump of selected main storage areas is required.

LIST=
identifies specific areas to be dumped.

TERMINAL
indicates that all storage areas associated with the terminal are to be dumped. These storage areas are as follows:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)
2. Common system area (CSA), including the user's portion of the CSA (CWA)
3. Trace table
4. All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task
5. Contents of general-purpose registers upon entry to dump control from the requesting task
6. Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped. The latter are used by basic mapping support.

PROGRAM
indicates that all program storage areas associated with this task are to be dumped. These storage areas include:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)
2. Common system area (CSA), including the user's portion of the CSA (CWA)
3. Trace table
4. All program storage areas containing user-written application program(s) active on behalf of the requesting task
5. Register save areas (RSAs) indicated by the RSA chain off the TCA

6. Contents of general-purpose registers upon entry to dump control from the requesting task

7. Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

TRANSACTION
is typically used in combination with other types of PARTIAL dump requests to include all transaction storage areas associated with the task. These areas include:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)

2. Common system area (CSA), including the user's portion of the CSA (CWA)

3. Trace table

4. Contents of general-purpose registers upon entry to dump control from the requesting task

5. All transaction storage areas chained off the TCA storage accounting field

6. Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

SEGMENT
is used to include in the PARTIAL dump any area of main storage specified (see Programming Note). In addition to the selected area, the contents of the following storage areas are displayed:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)

2. Common system area (CSA), including the user's portion of the CSA (CWA)

3. Trace table

4. Contents of general-purpose registers upon entry to dump control from the requesting task

5. Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

These parameters are not mutually exclusive. They can be specified in any combination and any order. The parentheses

are optional when only one parameter is specified. At least one parameter is required. No storage area is dumped more than once as a result of a single DFHDC TYPE=PARTIAL request. Thus, for example, if DFHDC TYPE=PARTIAL,LIST=(TERMINAL,TRANSACTION) is specified, the contents of the TCA and CSA are displayed only once.

DMPCODE=
indicates that a four-character dump code is to be printed out with the dump to identify it; this code should be unique so that it is informative concerning the condition that caused the dump.

value
is a combination of four alphabetic, numeric, and/or special characters to be printed as the dump code.

YES
indicates that the dump code has been placed in TCADCDC.

Programming Note:

If SEGMENT is specified, the application programmer must code two instructions that place the address of the main storage area to be dumped into TCADCSA and the length (in binary) of the area to be dumped into TCADCNB prior to execution of the DFHDC TYPE=PARTIAL macro instruction. The maximum length that can be specified in TCADCNB is 32,767 bytes. The specified area must be a valid area, that is, storage allocated by the operating system within the CICS/VS region/partition boundaries.

Examples

The following example illustrates the coding required to request a PARTIAL storage dump that includes, along with all program storage areas, all transaction storage areas associated with this task:

```
DFHDC TYPE=PARTIAL,          REQUEST PARTIAL STORAGE DUMP       *
      LIST=(TRANSACTION,     AREAS ASSOCIATED WITH TRANSACTION  *
      PROGRAM),              PROGRAM STORAGE AREAS              *
      DMPCODE=DT&P           USER-SPECIFIED DUMP CODE
```

The preceding example is applicable to Assembler language, ANS COBOL, or PL/I programs. All values passed to CICS/VS are specified in the DFHDC macro instruction. As noted above, when SEGMENT is specified, certain values must be stored in fields of the TCA prior to execution of the DFHDC macro instruction. The programmer can also store the dump code in the TCA prior to execution of the macro instruction. Thus, the following examples show how to request a PARTIAL dump of a selected main storage area, using either Assembler language, ANS COBOL, or PL/I.

For Assembler language:

```
ST    R5,TCADCSA             PLACE STORAGE ADDRESS IN TCA
MVC   TCADCNB,=H'16384'      PLACE LENGTH OF AREA IN TCA
MVC   TCADCDC,=CL4'AB12'     PLACE DUMP CODE IN TCA
DFHDC TYPE=PARTIAL,          REQUEST PARTIAL STORAGE DUMP       *
      LIST=SEGMENT,          DUMP AREA PREVIOUSLY SPECIFIED     *
      DMPCODE=YES            DUMP CODE PREVIOUSLY SPECIFIED
```

For ANS COBOL:

```
MOVE R5 TO TCADCSA.          NOTE PLACE STRG ADDRESS IN TCA.
MOVE 16384 TO TCADCNB.       NOTE PLACE LENGTH OF AREA IN TCA.
```

```
MOVE 'AB12' TO TCADCDC.     NOTE PLACE DUMP CODE IN TCA.
DFHDC TYPE=PARTIAL,         REQUEST PARTIAL STORAGE DUMP       *
      LIST=SEGMENT,         DUMP AREA PREVIOUSLY SPECIFIED     *
      DMPCODE=YES           DUMP CODE PREVIOUSLY SPECIFIED
```

For PL/I:

```
TCADCSA=R5;                 /*PLACE STORAGE ADDRESS IN TCA*/
TCADCNB=16384;              /*PLACE LENGTH OF AREA IN TCA*/
TCADCDC='AB12';             /*PLACE DUMP CODE IN TCA*/
DFHDC TYPE=PARTIAL,         REQUEST PARTIAL STORAGE DUMP       *
      LIST=SEGMENT,         DUMP AREA PREVIOUSLY SPECIFIED     *
      DMPCODE=YES           DUMP CODE PREVIOUSLY SPECIFIED
```

CHAPTER 9. CICS/VS BUILT-IN FUNCTIONS

Several commonly used functions are available to the application programmer through CICS/VS macro instructions. These are functions which, in the past, have generally had to be coded as separate subroutines by the programmer. These functions, referred to as built-in functions, provide the following services:

- Table search
- Phonetic conversion
- Verification of a data field
- Editing of a data field
- Bit manipulation
- Input formatting
- Weighted retrieval

Requests for these services are communicated to the CICS/VS built-in functions program (DFHBFP) through DFHBIF macro instructions. DFHBFP then executes, at the priority of the requesting program, under control of the requesting program's TCA common control communication area (TCACCCA). Normally, control is returned to the next sequential instruction following the macro expansion in the requesting program; however, the application programmer can specify conditional branch options in the macro request if desired. Since DFHBFP uses the TCACCCA, the application program must issue the DFHBFTCA macro instruction to copy the symbolic storage definition for the DFHBFP TCACCCA and store any required information therein before issuing the DFHBIF macro instruction.

The available DFHBFP services are summarized briefly in the following paragraphs. The formats and parameters of DFHBIF macro instructions used to request these services are explained in detail in the reference portion of this chapter.

TABLE SEARCH

The table search built-in function provides the application programmer with the means of conveniently searching a table for a particular entry and having a corresponding value within that table or a second table, the address of the corresponding value, and the index of the selected entry (relative to one) returned. A specified search argument is compared on a byte-for-byte basis against a specified field of entries in the table being searched. The programmer can elect to have a default value returned in lieu of a corresponding value if the searched-for entry is not found. If an index was requested, but the entry is not found, an index value of zero is returned to the application program.

PHONETIC CONVERSION

The phonetic conversion built-in function provides the application programmer with the capability of converting a name into a key, which can then be used to access data in a data base data set. The key that is generated is based upon the phonetic sound of the name; hence, names that sound alike, even though spelled differently, produce identical keys. For example, the names SMITH, SMYTH, and SMYTHE produce a phonetic key of S530. In addition to the phonetic conversion built-in function, a CICS/VS subroutine that performs similar conversion of keys is available for use by offline user-written programs. Together, these

facilities allow the CICS/VS user to organize his file according to name (or any similar alphabetic key) and access the file using search arguments that may be misspelled or misunderstood to retrieve required data.

VERIFICATION OF A DATA FIELD

The field verify function enables the application programmer to determine whether the contents of a data field is:

- Entirely alphabetic: blanks or A-Z
- Entirely EBCDIC numbers, with or without trailing sign: 0-9 (X'F0' through X'F9')
- Entirely packed decimal (COMPUTATIONAL-3 in American National Standard (ANS) COBOL or FIXED DECIMAL in PL/I)

A branch is made to an appropriate user-written routine accordingly.

EDITING OF A DATA FIELD

The field edit function is a means of removing alphabetic and/or special characters from an EBCDIC data field. The remaining, numeric characters are right-justified with zero padding at the left as necessary. If the field ends with a minus sign or 'CR', a negative zone is placed over the low-order byte.

BIT MANIPULATION

The bit manipulation function enables the application programmer to check specific bits in a byte for a bit-on or bit-off condition, or to change the state of specific bits in a byte. The application programmer specifies the eight-bit mask (bit pattern) to be applied against the byte containing bits to be operated on. The mask can be described by a single EBCDIC character within single quotation marks: for example, '5' or 'M'. Alternatively, the symbolic address of a one-byte field containing the mask can be specified. If desired, control can be transferred to a specified location if all affected bits (or all bits in the byte) are on, or if all affected bits (or all bits in the byte) are off, after completion of the bit manipulation. This function is particularly useful for ANS COBOL programs, which are otherwise unable to carry out bit manipulation.

INPUT FORMATTING

The input formatting function is provided to assist the application programmer in designing his program to handle free-format input entered from a terminal. Input formatting allows the application programmer to convert the free-format terminal input into a predefined fixed format to be processed by the application program. By means of this function, the application program can be coded to accept any of three forms of input from the terminal. These forms are discussed in order of increased flexibility below.

FIXED FORMAT

This is the simplest case, but it requires a rigid adherence to form. The input transaction must be identical in format to the fixed internal format established by statements in the application program.

For example, assume the fixed internal format for data consisting of a transaction identification; last name, first name, and middle initial; and identification number is as follows:

Cols	1-4	5	7	19	21	31	33	34	36	42
	TRANS ID	ØØ	LAST NAME	ØØ	FIRST NAME	ØØ	M I	ØØ	ID NO	EOB

The input transaction must be formatted as shown by the following example:

```
INQR     HUGHES       JOHN          Q       096556   EOB
```

Each input field must be entered with the terminal operator in the positions established for it in the fixed internal format.

POSITIONAL FORMAT

This option allows the terminal operator to enter a system-programmer selected field-separator character to indicate the end of a field or the absence of a field. The order of the fields on input must be the same as the order established for the fixed internal format; the field lengths need not be the same. If other fields follow, the end of a field or the absence of a field within the input must be indicated by a field-separator character.

Assume that input consisting of a transaction identification; last name, first name, and middle initial; and identification number is to be entered from a terminal. Further assume that the input formatting function is invoked by the application program to process this input, recognizing the slash (/) as a field-separator character. The following examples show permissible free-format positional input. Each input transaction is terminated by an end-of-block (EOB) character.

- Complete input
  INQR/HUGHES/JOHN/Q/096556 EOB
- Middle initial unknown
  INQR/HUGHES/JOHN//096556 EOB
- Middle initial and identification number unknown
  INQR/HUGHES/JOHN EOB

KEYWORD FORMAT

The keyword format provides an even greater degree of flexibility in that terminal input can be entered in any order. The terminal operator need only be familiar with the keyword identifiers that the application programmer has established for the input fields. Each keyword identifier consists of up to four characters selected by the application programmer. The keyword identifier must be preceded by a one-character field-name start character and followed by a one-character field-separator character, both of which must be specified at system initialization by the system programmer.

As an example, assume that keyword identifiers are established as follows:

```
LN  -  last name field
FN  -  first name field
MI  -  middle initial field
ID  -  identification number
```

Further assume that the period has been specified as the field-name start character and the equal sign has been specified as the field-separator character. The following examples show permissible free-format keyword input.

- Complete input
  ```
  INQR.FN=JOHN.MI=Q.ID=096556.LN=HUGHES EOB
  ```
- First name unknown
  ```
  INQR.LN=HUGHES.MI=Q.ID=096556 EOB
  ```
- First name and identification number unknown
  ```
  INQR.LN=HUGHES.MI=Q EOB
  ```

The first entry in each of these examples is the transaction identification. Since CICS/VS expects this identification, it must be first and no keyword identifier is required for it. Succeeding data fields are entered in any order. The input is terminated by the end-of-block (EOB) character.

COMBINATION INPUT

CICS/VS DFHBIF macro instructions can be included in an application program to permit a combination of fixed, positional, and keyword input. For example, the following variations may be allowed.

- Complete input
  ```
  INQR.LN=HUGHES.FN=JOHN/Q/096556 EOB
  INQR.FN=JOHN.LN=HUGHES.MI=Q/096556 EOB
  INQR/HUGHES/JOHN/Q  096556 EOB
  ```
- First name unknown
  ```
  INQR.LN=HUGHES//Q/096556 EOB
  INQR/HUGHES//Q.ID=096556 EOB
  INQR  HUGHES//Q  096556 EOB
  ```
- First name and identification number unknown
  ```
  INQR.LN=HUGHES//Q EOB
  INQR/HUGHES.MI=Q EOB
  INQR  HUGHES.MI=Q EOB
  ```

WEIGHTED RETRIEVAL

The weighted retrieval function allows the application programmer to search a group of records on a VSAM data set, selecting only those records that satisfy or are closest to selection criteria he provides. Each record with a specified partial key (beginning with the first one, or with the one having a specified relative number) is examined to see whether entries in certain other fields of the record match the values specified for those fields as selection criteria. Matching may be on exact comparison or within a given range.

Differentiating among individuals is one example of an area in which weighted retrieval processing is advantageous. In federal and state governments, the banking industry, and many other application areas dealing with large populations, name alone does not provide unique identification. A number of people may have the same name, so additional identifying characteristics are required. Such attributes as sex, race, birth date, address, relatives, and employment tend to permit unique identification. A basic example showing weighted retrieval on the basis of last name, first initial, and mother's maiden name is given in the reference section of this chapter, after the detailed formats of the DFHBIF macro instructions for weighted retrieval have been described.

Each comparison that is performed during weighted retrieval processing causes a match value to be added to a current total counter maintained automatically by CICS/VS. If the comparison yields a match, the match value is also added to a weighted counter. If the compared fields do not match, the no-match value is subtracted from the weighted counter. Fields in the search criteria or in a record being examined that contain a predefined null character may be ignored (not included in the search) if desired.

When all fields to be considered in the selection process have been examined, a weighted qualification percentage (WQP) is calculated for the record. If this percentage is within the limits of acceptability established in the application program, the percentage and complete key of the record are saved in a key-save storage block.

After all records with the specified partial key have been examined, the saved keys are sorted into descending percentage-value order. Under normal processing, the records whose keys have been saved are retrieved one at a time and made available to the application program in order of decreasing acceptability. A further judgment as to acceptability or verification of identifiers is then made by the application program, which may involve the terminal operator in the final selection process.

If the number of saved keys exceeds a maximum established in the application program (say, n), all keys having a weighted qualification percentage (WQP) equal to or lower than that of the "n+1"th key are dropped. If this dropping causes less than the application program-specified maximum number of keys to be saved but some keys are saved (as in Figure 9-1), no indication is given to the application program. However, if all percentages are the same so that all keys are dropped thereby, control is passed to an overflow routine (if one is specified in the application program). If the amount of storage required for saved keys exceeds the amount of storage available for keys, an overflow also occurs, and the application program is notified. An alternative, lower maximum can be established by the application program. The maximum number of records that can be retrieved is restricted by the maximum size of a key-save block (64K). This maximum is calculated as storage size divided by saved key length plus one.

NRECDS=maximum number of records to be made available to application program, say, N

INPUTNO=maximum number of records to be evaluated from the data set, say, I

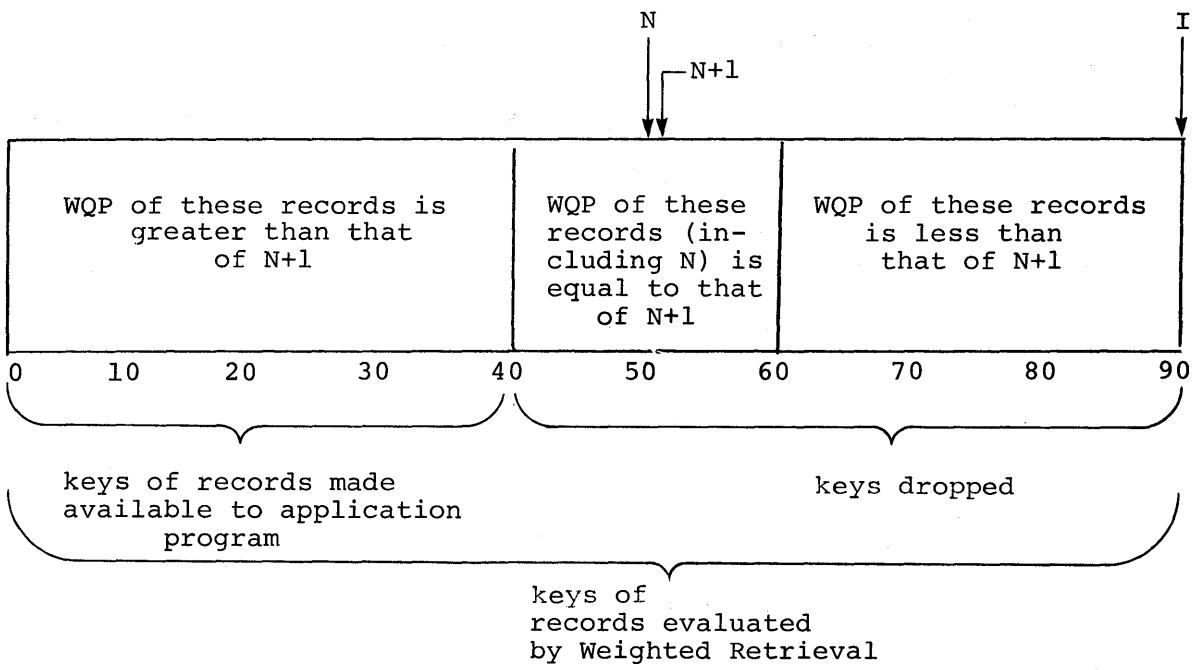


Figure 9-1 Selection of Records by Weighted Retrieval

Notes:

1. Because of the potential effect of weighted retrieval operations on system performance, this function should not be used indiscriminately. The amount of file accessing and the use of internal storage should be taken into account.

2. The computations applied by CICS/VS in weighted retrieval processing can be expressed as follows:

   Let MV = match value
   NMV = no-match value

   a. The weighted counter (WC), which holds the sum of all match values that had a match minus the sum of all no-match values that had no match:

      WC = (MV + MV + ... + MV) - (NMV + NMV + ... + NMV)

   b. The sum of all match values specified in WTRTPARM macro instructions for the weighted retrieval operation; the potential count (PC):

      PC = MV + MV + ... MV

c. The sum of all match values generated by the record comparisons (excludes those comparisons bypassed because the null character is present); the current total counter (CTC):

$$CTC = MV + MV + \ldots + MV \qquad n \leq k$$

d. The weighted potential (WP):

$$WP = \frac{PC + CTC}{2}$$

e. the weighted qualification percentage (WQP):

$$WQP = \frac{WC}{WP}$$

An overall effect of this method of computation is to provide a minimum weighting penalty for records having absent fields but yet prevent them from being chosen in preference to records that have all identifiers present.

DFHBFTCA MACRO INSTRUCTION

When CICS/VS built-in functions (BIF) are utilized in an application program, the symbolic storage definition for the TCACCCA used by BIF must be copied in the application program. This copying is achieved by means of a DFHBFTCA macro instruction, which must immediately follow the statement that copies the TCA and the user's definition of a TWA, if any. The general format of this macro instruction is as follows:

```
┌──────┬────────┬──────────────────────────────────────────────────────┐
│      │        │                                                      │
│      │DFHBFTCA│ [OPTION={BASIC}]                                     │
│      │        │         {WTRET}                                      │
│      │        │                                                      │
└──────┴────────┴──────────────────────────────────────────────────────┘
```

where:

OPTION=
indicates which BIF are to be utilized.

BASIC
is required if any of the following functions are requested: table search, phonetic conversion, field verify, field edit, bit manipulation, or input formatting.

WTRET
is required if weighted retrieval is utilized.

One of these parameters may be specified in the DFHBFTCA macro instruction. If the OPTION operand is omitted, both BASIC and WTRET are assumed.

Examples

Statements to copy the symbolic storage definitions referenced by BIF, positioned as required, are shown below.

For Assembler language:

```
         COPY  DFHTCADS
NAME     DS    CL20
STREET   DS    CL20
CITY     DS    CL10
STATE    DS    CL3
         DFHBFTCA
```

For ANS COBOL:

```
01 DFHTCADS COPY DFHTCADS.
   02 NAME PICTURE X(20).
   02 STREET PICTURE X(20).
   02 CITY PICTURE X(10).
   02 STATE PICTURE X(3).
   DFHBFTCA
```

For PL/I:

```
%INCLUDE (DFHTCADS);
  2 NAME CHAR(20),
  2 STREET CHAR(20),
  2 CITY CHAR(10),
  2 STATE CHAR(3);
  DFHBFTCA
```

TABLE SEARCH

The application programmer can search a table by issuing the following macro instruction:

```
DFHBIF  TYPE=TSEARCH
[,ARG=symbolic address]
[,TARGET=symbolic address]
[,ATABLE=([symbolic address1][,{symbolic address2}]
                               {YES            }
          [,numeric value1][,{numeric value2}]
                             {YES           }
          [,numeric value3])
[,FTABLE=([{symbolic address1}] [,{symbolic address2}]
           {YES               }  {YES               }
          [,{numeric value1}]  [,{numeric value2}])
            {YES           }    {YES           }
[,ORDER={ASCENDING }]
        {DESCENDING}
[,SUBST={symbolic address}] [,NOMATCH=symbolic address]
        {'literal value' }
[,INDEX=symbolic address]
[,RANGE=YES]
[,ERROR=symbolic address]
```

where:

TYPE=TSEARCH
: indicates that the table search function is required.

ARG=symbolic address
: is the symbolic address of the field that contains the search argument; if omitted, the address is assumed to be in TCATSA1, a fullword field.

TARGET=symbolic address
: is the symbolic address of the field in which the function value is to be returned to the application program. The address of the function value is placed in TCATSA5, a fullword field, regardless of whether TARGET is specified.

ATABLE=
: is a description of the table to be searched (the argument table).

  symbolic address1
  : is the address of the first entry in the argument table; if omitted, the address is assumed to be in TCATSA2, a fullword field.

symbolic address2 or YES
is the address of the field in the first entry of the argument table to be compared with the search argument. If YES is specified, the field address is assumed to be in TCATSA4, a fullword field. If this operand entry is omitted, symbolic address2 is assumed to be the same as symbolic address1. If specified, the address represented by symbolic address2 must be equal to or greater than the address represented by symbolic address1. If it is not, bit 4 of TCATSRC is set on and no search is made.

numeric value1
is the length of each entry in the argument table (including any other fields in the entry or slack bytes required for boundary alignment). A value in the range from 1 through 32767 may be specified. If this operand entry is omitted, the length is assumed to be in TCATSH2, a halfword field.

numeric value2 or YES
is the length of the field in the argument table to be compared with the search argument. If YES is specified, the length is assumed to be in TCATSAF, a one-byte field. If this operand entry is omitted, numeric value2 is assumed to be the same as numeric value1. If specified, the value must be between 1 and 255 inclusive. If numeric value1 is not within this range, numeric value2 must be specified.

numeric value3
is the maximum number of entries to be searched. A value in the range from 1 through 32767 may be specified. If this operand entry is omitted, the numeric value is assumed to be in TCATSH1, a halfword field.

If one or more of these operand entries are omitted, but other operand entries follow, the comma that ordinarily follows an omitted entry must be included in the operand.

FTABLE=
is a description of the table from which a value is to be retrieved (the function table). If no function value is to be retrieved (for example, if only the index of a matching argument table entry is needed), this operand can be omitted. If this operand is specified, but some entries are omitted, the values of the corresponding entries in the ATABLE operand are assumed to apply.

If a complex table (where each table entry contains both an argument and a function value) is being searched, the argument table and function table, as defined for this macro instruction, are actually within the same table in storage. Alternatively, two separate tables--one containing search fields and one containing function values--may be used.

symbolic address1 or YES
is the address of the first function table entry. If YES is specified, the address is assumed to be in TCATSA3, a fullword field.

symbolic address2 or YES
is the address of the function field within the first function table entry. If YES is specified, the address is assumed to be in TCATSA5, a fullword field. This address must be equal to or greater than symbolic address1. If it is not, bit 5 of TCATSRPC is set on and no search is made.

numeric value1 or YES
is the length of each entry in the function table (including any other fields in the entry or slack bytes required for boundary alignment). A value in the range from 1 through 32767 may be specified. If YES is specified, the value is assumed to be in TCATSH3, a halfword field.

numeric value2 or YES
is the length of the field to be retrieved from the function table. If YES is specified, the length is assumed to be in TCATSFF, a one-byte field. The length must be between 1 and 255 inclusive. Note that if this operand is omitted, the default is the corresponding entry in the ATABLE, or its default if the corresponding entry is not specified in the ATABLE. The default for this operand is not numeric value1 above.

ORDER=
describes the sequence used in ordering the entries of the argument table and is optional if RANGE is not specified. The sequence must be EBCDIC; packed, fullword and halfword binary, and floating-point tables cannot be searched. When this parameter is specified, a quick binary search is used (rather than a sequential search).

ASCENDING
indicates that table entries are organized in ascending order according to the entries in the field to be compared with the search argument.

DESCENDING
indicates that table entries are organized in descending order according to the entries in the field to be compared with the search argument.

In either case, the field values are interpreted as EBCDIC representations. If this operand is not specified, the argument table is assumed to be unordered and is searched sequentially.

SUBST=
is an optional operand that specifies a value to be stored in TARGET if no entry matching the search argument is found in the argument table.

symbolic address
is the address of a field that contains the value to be stored.

'literal value'
is the value to be stored; single quotation marks must enclose the value in this specification but are not stored as part of the data.

If this operand is specified, the TARGET operand must be specified, and the NOMATCH operand cannot be specified.

INDEX=symbolic address
is an optional operand that identifies a halfword field in which an index value relative to one, identifying the matching argument-table entry, is to be returned to the application program. In addition, the index value is placed in TCATSH4, a halfword field, whether or not the INDEX operand is specified. Both fields contain zero if no matching entry is found.

RANGE=YES
is an optional operand indicating that, if no field compared with the search argument is an exact match, an existing table entry that would be adjacent to such an entry is to be taken as the function value. When this operand is specified, ORDER must be specified; otherwise, a sequential search of the table is made.

ERROR=symbolic address
is the address to which control is to be transferred if an error occurs. This branch is taken, for example, if the address specified for the function field to be examined is lower than the address specified for the first function table entry.

NOMATCH=symbolic address
is the address to which control is to be transferred if no table entry matching the search argument is found. If this operand is specified, the SUBST operand cannot be specified.

Returned Values

An entry in the argument table that matches the search argument satisfies the table search function. If such an entry is found, the address of the corresponding entry in the function table is returned in TCATSA5, a fullword field. If the TARGET operand is specified in the DFHBIF TYPE=TSEARCH macro instruction, the function value is returned in the location identified by that operand. If the function table contains more than one matching entry, the address (and the function value, if requested) of the first matching entry encountered during the search is returned. The index of the matching entry is returned in TCATSH4 and in the field identified by the INDEX operand if specified.

If the RANGE=YES operand is specified, a matching entry satisfies the search as described above. If no matching entry is found, the search is satisfied in an alternative manner:

- If ORDER=ASCENDING is specified, the argument table entry having the largest argument value less than the search argument satisfies the search.
- If ORDER=DESCENDING is specified, the argument table entry having the smallest argument value greater than the search argument satisfies the search.

Figure 9-2 defines the conditions that may occur during a table search and defines the possible return codes.

Condition	Response Code		
	Assembler	ANS COBOL	PL/I
Match Found	X'00'	12-0-1-8-9 (TCATSMH)	00000000
ATABLE Field Address < Entry Address (symbolic address2 < symbolic address1)	X'04'	12-4-9 (TCATSER2)	00000100
FTABLE Field Address < Entry Address (symbolic address2 < symbolic address1)	X'08'	12-8-9 (TCATSER1)	00001000
No Match Found	X'F0'	'0'	'0'

Note: The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.

Figure 9-2. Table Search Response Codes

Example - Separate Tables

The following example shows how the DFHBIF TYPE=TSEARCH macro instruction can be used in an Assembler-language program. A four-character argument is matched against fields in a seven-entry argument table. If the search is satisfied, the address of a two-character corresponding field in the function table is placed in TCATSA5 and the index value of the matching entry is placed in TCATSH4. If no matching entry is found, a branch to BR1 occurs.

```
        DFHBIF  TYPE=TSEARCH,                                          *
                ARG=ARG1,                                              *
                ATABLE=(ATBL,AFLD,9,4,7),                              *
                FTABLE=(FTBL,FFLD,5,2),                                *
                ERROR=ERROR1,                                          *
                NOMATCH=BR1
          .
          .
ERROR1
          .
          .
BR1
          .
          .
ATBL    DS  0XL9          FIRST ENTRY OF ARG TABLE
        DS  XL5
AFLD    DS  XL4           FIRST ARGUMENT FIELD
        DS  6XL9          SPACE FOR SIX MORE ENTRIES
FTBL    DS  0XL5          FIRST ENTRY OF FUN TABLE
        DS  XL3
FFLD    DS  XL2           FIRST FUNCTION FIELD
        DS  6XL5          SPACE FOR SIX MORE ENTRIES
ARG1    DS  XL4           SEARCH ARGUMENT
          .
          .
```

Example - Complex Table

Both the argument values and the function values may be in the same table, in which case, the table is referred to as a complex table. The search is similar to that above, except that only one table is described.

```
       DFHBIF  TYPE=TSEARCH,                                          *
               ARG=ARG1,                                              *
               ATABLE=(TBL1,FLDA,5,2,3),                              *
               FTABLE=(TBL1,FLDF,5,3),                                *
               ERROR=ERROR1,                                          *
               NOMATCH=BR1
         .
         .
ERROR1
         .
         .
BR1
         .
         .
TBL1   DS  0CL5          FIRST ENTRY OF ARG/FUN TABLE
FLDA   DS  CL2           FIRST ARGUMENT FIELD
FLDF   DS  CL3           FIRST FUNCTION FIELD
       DS  2CL5          SPACE FOR TWO MORE ENTRIES
ARG1   DS  CL2           SEARCH ARGUMENT
         .
         .
```

PHONETIC CONVERSION

The application programmer can request that a 16-byte field of data be phonetically encoded by issuing the following macro instruction:

```
┌-------------------------------------------------------------------┐
|       |        |                                                  |
|       | DFHBIF |  TYPE=PHONETIC                                   |
|       |        |[,FIELD=symbclic address]                         |
|       |        |[,ERROR=symbclic address]                         |
|       |        |                                                  |
└-------------------------------------------------------------------┘
```

where:

TYPE=PHONETIC
indicates that the phonetic Conversion function is required.

FIELD=symbolic address
is the symbolic address of the 16-byte field containing the data to be phonetically encoded. The first character of this field must be alphabetic (one of the letters from A through Z, in either upper- or lowercase). If it is not, TCAPHNR, a one-byte field, is set to X'50' and no phonetic conversion is made. If this operand is omitted, the data to be converted is assumed to be in TCANAME, a 16-byte field.

Note: For ANS COBOL, TCAPHNR may be tested for X'50' by using the 88-level definition (TCAPINN) provided by CICS/VS.

ERROR=symbolic address
is the symbolic address to which control is transferred if the first character of the 16-byte field to be converted is nonalphabetic.

Returned Value

The returned value is placed at TCAPHON. This value is the four-byte phonetic equivalent of the data passed to the function. It consists of the first character of the data and three EBCDIC numbers representing the characters in the remainder of the data.

Phonetic Coding Method

The application programmer need not be familiar with the CICS/VS method of phonetic coding to use the phonetic conversion function. It is summarized here for informational purposes only. Remember that the first character of the field to be coded is not changed; it becomes the first character of the returned value. Three digits are selected to represent the remaining characters in accordance with the following rules:

Original Character	Code Value
B, P, F, V	1
C, G, J, K, Q, S, X, Z	2
D, T	3
L	4
M, N	5
R	6
A, E, H, I, O, Y, W, U, blanks, and nonalphabetic characters	Bypassed, no code value

- Lowercase letters are translated to uppercase for purpose of conversion.
- Double letters are coded as a single letter.
- Two or more adjacent letters with the same value are coded as a single letter.
- If more than three EBCDIC numbers can be computed from the data, only the first three are used.
- If fewer than three numbers can be computed from the name, the result is padded on the right with EBCDIC zeros to form a four-byte result.

Examples

```
DFHBIF  TYPE=PHONETIC,                                          *
        FIELD=NAME
```

where NAME is a 16-byte field, yields results as follows:

```
LEHMICKE    yields    L520
WONG        yields    W520
SOO         yields    S000
```

Phonetic Conversion Subroutine

A CICS/VS subroutine that performs phonetic conversion of 16-character names in the same manner as the phonetic conversion built-in function is available for use by offline user-written programs. The subroutine can be called by a program running under any of the operating systems under which CICS/VS can be run. A 16-character name

to be converted is provided as input to the subroutine; the four-byte phonetic equivalent of that name is returned as a result. The rules given above under "Phonetic Coding Method" are applied in the conversion process.

The general forms of the macro instruction to invoke the subroutine are as follows:

For Assembler language:

```
CALL DFHPHN,(lang,name,phon)
```

For ANS COBOL:

```
CALL 'DFHPHN' USING lang name phon.
```

For PL/I:

```
CALL DFHPHN (lang,name,phon) ;
```

where:

lang
: is the symbolic address of a one-character code indicating the programming language being used: X'F0' indicates ANS COBOL or Assembler language; X'F1' indicates PL/I. If an error occurs during processing of this request, X'50' is returned in this location. If no error occurs, X'00' is returned, and the location must be reset to indicate the programming language before the location can be reused.

name
: is the symbolic address of the field that contains the 16-character name to be converted.

phon
: is the symbolic address of the field in which the four-byte phonetic equivalent of the name passed to the subroutine is returned to the calling program.

FIELD VERIFY

The application programmer can check whether a field is entirely alphabetic, EBCDIC numeric, or packed decimal (COMP-3 in ANS COBOL or FIXED DECIMAL in PL/I) by issuing the following macro instruction:

```
DFHBIF  TYPE=FVERIFY
       [,FIELD=symbolic address]
       [,LENGTH={symbolic address}]
                {numeric value   }
       [,ALPHA=symbolic address]
       [,NUMERIC=symbolic address]
       [,PACKED=symbolic address]
```

where:

TYPE=FVERIFY
indicates that the field verify function is required.

FIELD=symbolic address
is the symbolic address of the field to be verified; if omitted, the field address is assumed to be in TCACKFD, a fullword field.

LENGTH=
gives the length, in bytes, of the field to be verified.

symbolic address
is the address of a halfword field containing the length of the field to be verified.

numeric value
is the length of the field to be verified.

The maximum length that can be specified is 32767. If this operand is omitted, the length is assumed to be in TCACKLN, a halfword field.

ALPHA=symbolic address
is the address to which control is transferred if the field consists entirely of alphabetic characters (A through Z) and/or blanks.

NUMERIC=symbolic address
is the address to which control is transferred if the field consists entirely of EBCDIC numbers (X'F0' through X'F9') with an optional trailing minus sign or 'CR'.

PACKED=symbolic address
is the address to which control is transferred if the field consists entirely of packed decimal characters, that is, of half-bytes with hexadecimal values from 0 through 9, except for the rightmost half-byte, which must contain a hexadecimal C, D, E, or F.

The ALPHA, NUMERIC, and PACKED operands may be specified in any combination or order, but at least one of them must be specified. The conditions specified are tested upon request in the order ALPHA, NUMERIC, PACKED, irrespective of the order of the operands. If none of the test conditions is met, control goes to the instruction following the DFHBIF TYPE=FVERIFY macro instruction in the application program.

Returned Values

The purpose of the field verify function is to determine what kind of data must be processed and cause control to be transferred to an appropriate routine in the application program accordingly. The application programmer can test the results of the verify function by testing the response code at TCACHKR. Figure 9-3 indicates the conditions and response codes.

Condition	Response Code		
	Assembler	ANS COBOL	PL/I
Packed field	X'20'	11-4-9 (TCACKPK)	00100000
Numeric field	X'40'	No punches (TCACKNM)	01000000
Alphabetic field	X'80'	12-0-1-8 (TCACKAL)	10000000
Mixed field	X'E0'	0-2-8 (TCACKMX)	11100000

Note: The names in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.

Figure 9-3. Field Verify Function Response Codes

Example

```
DFHBIF  TYPE=FVERIFY,                                          *
        FIELD=CONT,                                            *
        LENGTH=16,                                             *
        ALPHA=MYROUT
```

Execution of the DFHBIF macro instruction above causes the contents of CONT, a 16-byte field, to be checked to determine whether it contains only alphabetic characters and/or blanks. If it does, control is transferred to MYROUT. Otherwise, control returns to the instruction following this DFHBIF instruction in the application program.

FIELD EDIT

The application programmer can request that alphabetic and/or special characters be removed from an EBCDIC data field by issuing the following macro instruction:

```
┌──────┬──────┬──────────────────────────────────────────────────┐
│      │      │                                                  │
│      │DFHBIF│  TYPE=DEEDIT                                     │
│      │      │[,FIELD=symbolic address]                         │
│      │      │[,LENGTH={symbolic address}]                      │
│      │      │[       {numeric value   }]                       │
│      │      │                                                  │
└──────┴──────┴──────────────────────────────────────────────────┘
```

where:

TYPE=DEEDIT
indicates that the field edit function is required.

FIELD=symbolic address
is the symbolic address of the field to be edited; if omitted, the field address is assumed to be in TCAFLD, a fullword field.

LENGTH=
gives the length, in bytes, of the field to be edited.

symbolic address
is the address of a halfword field containing the length of the field to be edited.

numeric value
is the length of the field to be edited.

The maximum length that can be specified is 32767. If this operand is omitted, the length is assumed to be in TCAFLN, a halfword field.

Returned Values

All bytes (except the rightmost byte) containing other than EBCDIC numeric characters are deleted from the data field. The remaining characters are right-justified in the field with zero padding at the left as necessary. If the field ends with a minus sign or a 'CR', a negative zone (X'D') is placed over the rightmost (low-order) byte. The zone portion of the rightmost byte may contain any hexadecimal character from X'A' through X'F'. The digit portion of this byte may contain one of the hexadecimal digits from X'0' through X'9'. Where this is the case, the rightmost byte is returned unaltered (see the example below). This permits the application program to operate on a zoned numeric field. In any case, the returned value is in the field that initially contained the unedited data.

Example

```
DFHBIF  TYPE=DEEDIT,                                          *
        FIELD=CONTG,                                          *
        LENGTH=9
```

Execution of this macro instruction removes all characters other than EBCDIC numbers from CONTG, a nine-byte field, and returns the edited result in that field to the application program. Say, for example, the field contains 14-6704/B before the DFHBIF TYPE=DEEDIT macro instruction is issued. After editing, it contains 00146704B.

BIT MANIPULATION

The application programmer can test or change the state of specific bits in computer storage by issuing the following macro instruction:

```
DFHBIF [TYPE={BITSETON }]
       [     {BITSETOFF}]
       [     {BITFLIP  }]
       [     {BITEST   }]
       [,FIELD=symbolic address]
       [,BIT={symbolic address}]
       [     {value           }]
       [,BITON=symbclic address]
       [,BITOFF=symbolic address]
```

where:

TYPE=
indicates that the bit manipulation function is requested and identifies the action required.

BITSETON
ensures that all bits selected by a specified bit pattern are on after execution of this macro instruction.

BITSETOFF
ensures that all bits selected by a specified bit pattern are off after execution of this macro instruction.

BITFLIP
causes the state of each bit selected by a specified bit pattern to be changed.

BITEST
causes the state of each bit selected by a specified bit pattern to be tested and an indicator to be set accordingly.

FIELD=symbolic address
is the symbolic address of the byte containing bits to be operated on; if omitted, the address is assumed to be in TCABITF, a fullword field.

BIT=
specifies the bit pattern (mask) to be applied to the specified byte.

symbolic address
is the address of a byte that contains the bit pattern.

value
is a single EBCDIC character enclosed in single quotation marks.

If this operand is omitted, the bit pattern is assumed to be in TCABITV, a one-byte field.

BITON=symbolic address
is the symbolic address to which control is transferred if--

- For BITSETON, BITSETOFF, or BITFLIP:
  All bits in the specified byte are on after the operation is completed

- For BITEST:
  All bits that are on in the bit pattern are on in the field that is tested

BITOFF=symbolic address
is the symbolic address to which control is transferred if--

- For BITSETON, BITSETOFF, or BITFLIP:
  All bits in the specified byte are off after the operation is completed

- For BITEST:
  All bits that are on in the bit pattern are off in the field that is tested

Returned Values

For BITSETON, BITSETOFF, or BITFLIP, the returned value is the contents of the byte specified in the FIELD operand, with selected bits modified as required. For BITEST, the result of the test is returned in TCABITR as shown in Figure 9-4. If BITON, BITOFF, or both BITON and BITOFF are specified, and if certain conditions are met as described in the explanations of these operands, control is transferred.

Condition	Response Code		
	Assembler	ANS COBOL	PL/I
Tested Bits Are Off	X'00'	12-0-1-8-9 (TCABIFOF)	00000000
Tested Bits Are On	X'F0'	'0' (TCABIFON)	'0'

Note: The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.

Figure 9-4. Bit Manipulation Response Codes

Examples

The macro instruction

```
            DFHBIF  TYPE=BITSETON,                              *
                    FIELD=DATAF,                                *
                    BIT=PATERN,                                 *
                    BITON=BLABEL
            .
            .
            .
     PATERN DC X'FF'
```

ensures that all bits of the one-byte field DATAF are set on and causes a branch to BLABEL.

The macro instruction

```
        DFHBIF  TYPE=BITEST,                                    *
                BIT='A',                                        *
                BITOFF=CLABEL
```

causes a bit pattern of 11000001 to be applied to the one-byte field whose address is stored in TCABITF. If all tested bits are off, control is transferred to CLABEL.

INPUT FORMATTING

The application programmer can write a program to handle free-format input entered from a terminal. The free-format input may be either positional or keyword-oriented, or both, and may be entered in combination with fixed-format input, as explained earlier in this section under "input formatting." Examples are:

```
INQR/HUGHES/JOHN/Q/096556 EOB                   positional
INQR.FN=JOHN.MI=Q.ID=096556.LN=HUGHES EOB       keyword-oriented
```

Any task that issues DFHBIF macro instructions to provide input formatting must be attached to a terminal.

Storage Definition

As a first step in defining storage, the programmer must copy the CICS/VS control section of the terminal input/output area (TIOA) into his program. Definitions of the fields for which input data may be entered should follow the definition of the CICS/VS control section. For example, the Assembler-language programmer may write the following code:

```
          COPY   DFHTIOA
*                              BEGINNING OF TIOA
TIOAIN    DS     CL4           TRANS ID
TIOALN    DS     CL15          LAST NAME
TIOAFN    DS     CL9           FIRST NAME
TIOAMI    DS     CL1           MIDDLE INITIAL
TIOAID    DS     CL6           IDENTIFICATION
```

TIOADBA is the CICS/VS-established name representing the first byte of the user's section of the TIOA for Assembler language only. Succeeding names are application-programmer-selected identifiers of the input fields. (To review the instructions for copying of symbolic storage definitions, see preceding sections of this manual.)

DFHBIF TYPE=DEFLDNM Macro Instruction

If free-format keyword-oriented input is desired, the application programmer can issue a DFHBIF macro instruction of the following format to define the keywords that the terminal user may specify. If this macro instruction is used by an ANS COBOL programmer, it must appear in the Working Storage section of his program. It must appear with other data definitions in an Assembler-language or PL/I program. This macro instruction is not needed if only free-format positional input is to be handled by a program.

```
┌──────────────────────────────────────────────────────────────┐
│       │        │                                             │
│       │ DFHBIF │  TYPE=DEFLDNM                               │
│       │        │ ,NAMES=(keyword[,keyword,...])              │
│       │        │ ,LABEL=symbclic address                     │
│       │        │                                             │
└──────────────────────────────────────────────────────────────┘
```

where:

TYPE=DEFLDNM

indicates that this macro instruction provides definitions of field names that may appear as keywords in free-format input.

NAMES=(keyword[,keyword,...])
is a list of the field names that may appear as keywords entered by the terminal user to indicate which fields are to receive input data. Each keyword may be from one to four characters in length. Any combination of alphabetic, numeric, and/or special characters may be specified. The keywords must be specified in this macro instruction in the same order as the corresponding fields that will hold the data are defined in the fixed-format TIOA.

LABEL=symbolic address
is the label to be assigned to the list of field names. This label must be unique within the application program and may be from one to eight characters in length.

For example, a DFHBIF TYPE=DEFLDNM macro instruction defining keywords that the user can enter to refer to fields of the TIOA is:

```
DFHBIF  TYPE=DEFLDNM,                                          *
        NAMES=(TRAN,LN,FN,MI,ID),                              *
        LABEL=DEFI
```

Use of similar names within the DFHBIF macro instruction and the TIOA definition is wise programming practice, but not a requirement. Thus, the following macro instruction is also acceptable.

```
DFHBIF  TYPE=DEFLDNM,                                          *
        NAMES=(TR,LAST,FIR,MID,IDEN),                          *
        LABEL=MYIN
```

Required Delimiters

When providing free-format keyword-oriented input capabilities to terminal users, the application programmer, working with system programmers, must define a field-name start character and a field-separator character for the system before initialization. (See the CICS/VS System Programmer's Reference Manual for details.)

DFHBIF TYPE=INFORMAT Macro Instruction

Data entered as free-format input is read into a TIOA in the same manner as other data entered from a terminal. CICS/VS places the address of the TIOA into TCTTEDA (as it must be for a formatting operation). To provide for the formatting of this free-format input, the application programmer issues a DFHBIF TYPE=INFORMAT macro instruction. The DFHBIF TYPE=INFORMAT instruction should be issued immediately following the terminal control (DFHTC) macro instruction that causes data to be moved into the TIOA. to make sure that the address of the TIOA containing data to be formatted is in TCTTEDA.

The input formatting function retrieves the input data from the input TIOA and reformats that data in a CICS/VS-acquired TIOA as indicated in the application programmer's definition of the storage area. The address of the fixed-format TIOA is returned in TCTTEDA to the application program. The application programmer should establish addressability to this TIOA immediately, just as for any TIOA used in the program. (See the instructions for copying symbolic storage definitions in this manual.)

The format of this macro instruction is as follows:

```
DFHBIF  TYPE=INFORMAT
        ,FIELDS=(symbolic address [ ,symbolic address,... ])
       [,NAMES={symbolic address}]
               {YES            }
       [,LENGTH={symbolic address}]
                {numeric value   }
       [,ERROR=symbolic address]
```

where:

TYPE=INFORMAT
indicates that formatting of terminal input is required.

FIELDS=(symbolic address[,symbolic address,...])
are the labels of fields defined within the internal fixed-format TIOA to which the input data is to be transferred. The fields must be named in the order that they appear in the TIOA, and there must be a one-to-one correspondence between the field names in this macro instruction and the fields in the TIOA. The length of each field is determined by calculations based on the location represented by the symbolic address of the following field. Each field should be at least one byte in length. For positional input, each field for which data may be entered (that is, each position in the receiving area of storage) must be defined.

NAMES=
indicates that field names may be present as keywords in the input data stream.

symbolic address
is the LABEL parameter specified in a DFHBIF TYPE=DEFLDNM macro instruction in which the keywords that may be specified are defined.

YES
indicates that the label specified in the DFHBIF TYPE=DEFLDNM macro instruction defining the field names is in TCAINA2, a fullword field.

LENGTH=
specifies the size of the TIOA to be acquired for the internal fixed-format representation of the data. This length must be sufficient to accommodate all fields specified in the FIELDS operand of this macro instruction. It is used in determining the length of the last field.

symbolic address
is the address of a halfword field that contains the length value.

numeric value
is the length, in bytes, of the area required for the TIOA.

If this operand is omitted, the length is assumed to be a TCAINH1, a halfword field.

ERROR=symbolic address
is the symbolic address to which control is transferred if an error condition occurs (see "Returned Values" below).

Note: If the DFHBIF TYPE=INFORMAT macro instruction is issued immediately following the read instruction, the address of the TIOA containing the data to be formatted will be stored in TCTTEDA. If any intervening macro instructions are issued, the application programmer is responsible for saving and restoring the contents of TCTTEDA. For ANS COBOL, TIOABAR must be loaded before a DFHBIF, because the expansion will contain "Call DFHCBLI using fields."

Returned Values

The address of the fixed-format TIOA containing the reformatted data is available in TCTTEDA. This address must be loaded into TIOABAR, the base register for the area.

Certain error conditions may be detected during execution of the DFHBIF TYPE=INFORMAT macro instruction. In such cases, an error indication (response code) is returned to the application program in TCAINRC, a one-byte field. The error conditions that may occur and the response code for each are shown in Figure 9-5. For error conditions other than X'F4', no reformatted data is returned; that is, TCTTEDA does not contain the address of a fixed-format TIOA containing the reformatted data.

Condition	Response Code		
	Assembler	ANS COBOL	PL/I
No Error	X'00'	12-0-1-8-9 (TCAINNOE)	00000000
The input data does not contain field-name start or field-separator characters. (Such data may not be erroneous, if deliberately entered in this manner.)	X'20'	11-0-1-8-9 (TCAINALS)	00100000
The input data contains two field-name start characters with no field-separator character between them.	X'F1'	'1' (TCAINER1)	'1'
The input data contains an invalid name.	X'F2'	'2' (TCAINER2)	'2'
A field name is specified in the input data, but no DFHBIF TYPE=DEFLDNM macro instruction is contained in the application program.	X'F3'	'3' (TCAINER3)	'3'
The length of an input data field exceeds the internal field size defined	X'F4'	'4' (TCAINER4)	'4'

Note: The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.

Figure 9-5. INFORMAT Response Codes

Note: Application programmers and terminal operators should be aware that if fixed-format input is provided to an application program designed to accept free-format input, field overrun (X'F4') errors are apt to occur.

Examples

Assume the TIOA definition and the first DFHBIF TYPE=DEFLDNM macro instruction above. Further assume that the period has been established as the field-name start character and the equal sign and the slash as field-separator characters.

The free-format positional input

```
INQR/HUGHES/JOHN/Q/096556 EOB
```

can be processed by issuing the following macro instruction:

```
DFHBIF TYPE=INFORMAT,                                          *
       FIELDS=(TIOAIN,TIOALN,TIOAFN,TIOAMI,TIOAID)
```

The free-format keyword input

```
INQR.FN=JOHN.MI=Q.ID=096556.LN=HUGHES EOB
```

can be processed by issuing the following macro instruction:

```
DFHBIF TYPE=INFORMAT,                                          *
       FIELDS=(TIOAIN,TIOALN,TIOAFN,TIOAMI,TIOAID),            *
       NAMES=DEFI
```

A mixture of free-format positional and keyword input can be handled by this latter form of DFHBIF TYPE=INFORMAT macro instruction. For example,

```
INQR.LN=HUGHES//Q/096556 EOB
```

will be handled correctly.

WEIGHTED RETRIEVAL

The application programmer can request that a VSAM data set be scanned and those records meeting certain selection criteria be made available to the application program. In general, a series of DFHBIF macro instructions is involved.

1. A DFHBIF TYPE=WTRETST macro instruction indicates the start of a weighted retrieval operation.

2. One or more DFHBIF TYPE=WTRTPARM macro instructions provide the specifications to be used by CICS/VS in the weighted retrieval process.

3. One or more DFHBIF TYPE=WTRETGET macro instructions retrieve one or more selected records.

4. A DFHBIF TYPE=WTRETREL macro instruction releases the VSAM work area (VSWA) and other main storage used for the weighted retrieval process.

5. A DFHBIF TYPE=WTRETCHK macro instruction performs a check on the success of a phase of the weighted retrieval process.

Each of these macro instructions is discussed more fully below.

Note: Because of the potential effect of weighted retrieval operations on system performance, this function should not be used indiscriminately. The amount of file accessing and the use of internal storage should be taken into account.

Initiate Weighted Retrieval

To indicate the start of a weighted retrieval operation, the application programmer issues a DFHBIF macro instruction of the following format:

```
        |         |
        | DFHBIF  |  TYPE=WTRETST
        |         |[,DATASET=symbolic name]
        |         |[,RDIDADR=symbolic address]
        |         |[,INPUTNO={symbolic address}]
        |         |          {numeric value   }
        |         |          {YES             }
        |         |[,INPUTST={symbolic address}]
        |         |          {numeric value   }
        |         |          {YES             }
        |         |[,INPUTPC=([suboperand1][,suboperand2])]
        |         |[,NRECDS={symbolic address}]
        |         |         {numeric value   }
        |         |         {YES             }
        |         |[,NORESP=symbolic address]
        |         |[,DSIDER=symbolic address]
        |         |[,NOTOPEN=symbolic address]
        |         |[,NOTFND=symbolic address]
        |         |[,INVREQ=symbolic address]
        |         |[,IOERROR=symbolic address]
        |         |[,OFLOW=symbclic address]
        |         |[,ILLOGIC=symbolic address]
        |         |
```

where:

TYPE=WTRETST
indicates that a weighted retrieval operation is to be initiated.

DATASET=symbolic name
is the one- to eight-character data set identification of the VSAM data set from which the retrieval is to be made; if omitted, the data set identifier is assumed to be in TCAWTDI, an eight-byte field.

RDIDADR=symbolic address
is the symbolic address of the record identification field that contains the partial key of the record at which the data set is to be positioned prior to the retrieval process; if omitted, the address is assumed to be in TCAWTRI, a fullword field. (The format of the record identification field for a VSAM data set is described under "record identification Field" in Chapter 11.)

INPUTNO=
indicates the maximum number of records that can be examined. A value from 1 to 32767 may be specified.

symbolic address
is the address of a halfword field that contains the maximum number of records that can be examined.

numeric value
is a decimal numeral indicating the maximum number of records that can be examined.

YES
indicates that the maximum number of records to be examined has been placed in TCAWTH1, a halfword field.

If this operand is omitted, a default value of 512 is placed in TCAWTH1.

INPUTST=
indicates the number of records with the specified partial key to be skipped before examination of records begins. The maximum value that can be specified is 32767.

symbolic address
is the address of a halfword field that contains the relative number of the record that is to be examined first.

numeric value
is a decimal numeral indicating the relative number of the record that is to be examined first.

YES
indicates that the relative number of the desired record has been placed in TCAWTH2, a halfword field.

If this operand is omitted, a default value of 0 is placed in TCAWTH2. The weighted retrieval begins with the first record having the specified partial key.

INPUTPC=
specifies the percentages to be used by the weighted retrieval function to determine the limits of acceptability.

suboperand1
specifies the maximum percentage, a value from 0 to 100; this value can be indicated by the symbolic address of a halfword field containing the maximum value, a decimal numeral, or YES, which indicates that the value has been placed in TCAWTH3. If omitted, the maximum percentage is assumed to be 100.

suboperand2
specifies the minimum percentage, a value from 0 to 100; this value can be indicated by the symbolic address of a halfword field containing the minimum value, a decimal numeral, or YES, which indicates that the value has been placed in TCAWTH4. If omitted, the minimum percentage is assumed to be 0.

If the first suboperand is omitted, but the second is specified, the comma that otherwise follows the first suboperand must be included. If only one suboperand is given, it is assumed to be the first suboperand (the maximum percentage, 100).

NRECDS=
indicates the maximum number of records to be made available to the application program. A value from 1 to 32767 can be specified.

symbolic address
is the address of a halfword field that contains the maximum number of records.

numeric value
is a decimal numeral indicating the maximum number of records.

YES
indicates that the maximum number has been placed in TCAWGCNT, a halfword field.

If this operand is omitted, a default value of 200 is assumed.

NORESP, DSIDER, NOTOPEN, NOTFND, INVREQ, IOERROR, OFLOW, and ILLOGIC are used to test the response of CICS/VS to this request for initiation of the weighted retrieval function. These operands can be specified in this macro instruction or in a DFHBIF TYPE=WTRETCHK macro instruction. The meaning of each operand is discussed under "Test Response to a Request for weighted retrieval."

Returned Values

The address of a VSAM work area (VSWA) to be used by weighted retrieval throughout this series of weighted retrieval operations is returned in TCAWRAA, a four-byte field. Since any CICS/VS macro instruction issued within the application program may cause the contents of TCAWRAA to be changed, the application programmer should save this address. It must be restored in TCAWRAA prior to any subsequent DFHBIF macro instruction included in this series of weighted retrieval operations. A response code indicating how CICS/VS has responded to this request is returned in TCAWTRC, a one-byte field (see "Test Response to a Request for weighted retrieval").

Establish Selection Criteria

The application programmer passes selection criteria to CICS/VS by means of DFHBIF TYPE=WTRTPARM macro instructions. One of these macro instructions must be coded for each field that is to be examined during the selection process. Match and no-match values are established separately for each field. Then, during weighted retrieval processing, the applicable match and no-match values for examined fields of a record are used to determine a weighted qualification percentage for the record. The format of each DFHBIF TYPE=WTRTPARM macro instruction is as follows:

```
DFHBIF  TYPE=WTRTPARM
[,FIELD1=([symbolic address][,numeric value][,char]) ]
[,FIELD2=([symbolic address1][,symbolic address2]) ]
[,NULL={symbolic address}]
       {character value }
       {YES             }
[,MATCH={symbolic address}]
        {numeric value   }
[,NOMATCH={symbolic address}]
          {numeric value   }
[,RANGE=(suboperand1,suboperand2[,suboperand3]) ]
```

where:

TYPE=WTRTPARM
: indicates that return parameters for a weighted retrieval operation are being specified.

FIELD1=
: specifies the characteristics of the search field to be compared against a corresponding field in records of the data set on which the weighted retrieval function is to operate.

symbolic address
is the symbolic address of the field. If omitted, the address of the field is assumed to be in TCAWPA1, a four-byte field.

numeric value
is the length of the field in bytes and may range from 1 to 32767. If omitted, the length of the field is assumed to be in TCAWPH1, a halfword field.

char
is one character indicating the format of the data in the field as follows:

Character	Data Format
C	EBCDIC characters
Z	Zoned decimal numbers
P	Packed decimal numbers
H	Halfword binary
F	Fullword binary

If this parameter is omitted, the character is assumed to be in TCAWPB1, a one-byte field.

If one of these operand entries is omitted but succeeding operand entries follow, the comma that otherwise follows the entry must be included in the operand.

Notes:

1. The application programmer must ensure that the integrity of FIELD1 is not destroyed prior to the first DFHBIF TYPE=WTRETGET macro instruction. These values are used by the built-in functions program (DFHBFP) at that time. In particular, it is not advisable to utilize an area within a TIOA for this value.

2. The largest decimal number that can be contained in a zoned decimal (Z) or packed decimal (P) field cannot exceed 16 digits, including the sign.

FIELD2=
specifies the location of the data in the field of each record of the data set involved in the comparison with the search data in FIELD1.

symbolic address1
is the symbolic address (label) of the first byte of the storage area that will contain the record to be examined. If omitted, the address of the main storage area is assumed to be in TCAWPA3, a four-byte field.

symbolic address2
is the symbolic address (label) of the field within the storage area identified by symbolic address1 to be used in the weighted retrieval comparison. If omitted, the address of the field is assumed to be in TCAWPA4, a four-byte field.

If the first operand entry is omitted but the second is specified, the comma that otherwise follows the first entry must be included in the operand.

NULL=
specifies a one-byte "null character" which, it present in either FIELD1 or FIELD2, indicates that no comparison is to be performed.

symbolic address
is the symbolic address of a one-byte field containing the null character.

character value
is a single EBCDIC character within single quotation marks.

YES
indicates that the null character has been placed in TCAWPNL, a one-byte field.

The null character cannot be a binary zero (that is, X'00'); such a specification is ignored.

MATCH=
specifies a value to be added to the current total counter if the comparison is performed and to the weighted counter if the compared fields are equal. The value may range from -32768 through +32767.

symbolic address
is the symbolic address of a halfword field containing the value.

numeric value
is a decimal numeral in the range stated above.

If this parameter is omitted, the value is assumed to be in TCAWPH2.

Note: All match and no-match values specified for a weighted retrieval operation must have like signs.

NOMATCH=
specifies a value to be subtracted from the weighted counter if the compared fields are not equal. The value may range from -32768 through +32767.

symbolic address
is the symbolic address of a halfword field containing the value.

numeric value
is a decimal numeral in the range stated above.

If this parameter is omitted, the value is assumed to be in TCAWPH3.

Note: All match and no-match values specified for a weighted retrieval operation must have like signs.

RANGE=
indicates that the compared fields are to be considered equal if FIELD2 falls within a given range of FIELD1.

suboperand1
specifies the type of range used in the comparison. This entry can be a single character or YES, which indicates that the single character specifying type has been placed in

TCAWPTR. The valid characters are as follows:

Character	Type of Range
P	Percentage
U	Units
V	Value

suboperand2
specifies the upper limit, exceeding the value in FIELD1, which is to be considered a match. This entry can be a positive numeric value up to 32767 or YES, which indicates that the upper limit has been placed in TCAWPH4.

suboperand3
specifies the lower limit, below the value in FIELD1, which is to be considered a match. This entry can be a positive numeric value up to 32767 or YES, which indicates that the lower limit has been placed in TCAWPH5.

If suboperand3 is omitted, suboperand2 is assumed to apply both above and below the value in FIELD1. For example, if the value in FIELD1 is 165 and RANGE=(U,5) is specified, then any value from 160 through 170 is considered a match. If RANGE=(U,5,10) is specified, then any value between 155 and 170 is considered a match. If RANGE=(P,20) is specified, then any value between 132 and 198 {165*(1±20%)} is acceptable. If RANGE=(V,190,160), then any value between 160 and 190 is acceptable. If the data field contains EBCDIC characters (that is, C is specified in the FIELD1 operand), the RANGE operand is ignored.

Note: The upper bound and lower bound values are computed using the following formulas (where K is the value of FIELD1):

1. For P-type range, specified (P,x,y) or (P,x):

```
UB = K * (1 + x/100)          UB = K * (1 + x/100)
                       or
LB = K * (1 - y/100)          LB = K * (1 - x/100)
```

2. For U-type range, specified (U,x,y) or (U,x):

```
UB = K + x          UB = K + x
           or
LB = K - y          LB = K - x
```

3. For V-type range, specified (V,x,y):

```
UB = x

LB = y
```

Retrieve Selected Records

To retrieve one record that has been saved by the weighted retrieval function, the application programmer issues a DFHBIF TYPE=WTRETGET macro instruction. Before issuing the instruction, the programmer must ensure TCAWRAA contains the address of the VSAM work area (VSWA) used

in this series of weighted retrieval operations. The format of the DFHBIF TYPE=WTRETGET macro instruction is as follows:

```
┌-------------------------------------------------------------------┐
|       |        |                                                  |
|       | DFHBIF|  TYPE=WTRETGET                                    |
|       |        |[,NORESP=symbolic address]                        |
|       |        |[,ENDFILE=symbolic address]                       |
|       |        |[,NOTOPEN=symbolic address]                       |
|       |        |[,NOTFND=symbolic address]                        |
|       |        |[,INVREQ=symbolic address]                        |
|       |        |[,IOERROR=symbolic address]                       |
|       |        |[,OFLOW=symbclic address]                         |
|       |        |[,ILLOGIC=symbolic address]                       |
|       |        |                                                  |
└-------------------------------------------------------------------┘
```

where:

TYPE=WTRETGET
: indicates that retrieval of the next record saved by weighted retrieval (as ordered according to decreasing weighted qualification percentage) is desired.

NORESP, ENDFILE, NOTOPEN, NOTFND, INVREQ, IOERROR, OFLOW, and ILLOGIC (VSAM only)
: are used to test the response of CICS/VS to this request for record retrieval. These operands can be specified in this macro instruction or in a DFHBIF TYPE=WTRETCHK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for weighted retrieval."

Returned Values

One record that was saved as the result of weighted retrieval is returned to the application program. The address of this record is contained in VSWAREA within the VSWA provided by the WTRETST macro instruction. The length of the record is returned in VSWALEN.

In addition, the contents of several halfword fields are significant. TCAWGH1 contains the highest percentage of acceptability for this weighted retrieval operation, TCAWGH2 contains the lowest percentage of acceptability for this weighted retrieval operation, TCAWGH3 contains the percentage of acceptability of this record, and TCAWGH4 contains the number of records left to be presented to the user. After the first DFHBIF TYPE=WTRETGET macro instruction, TCAWGH5 contains a count of any records dropped to remain within the maximum specified in the NRECDS operand of the WTRETST macro instruction. On succeeding WTRETGETs, TCAWGH5 contains zero. The full key of the returned record is returned at the location specified in the RDIDADR operand of the DFHBIF TYPE=WTRETST macro instruction initiating this weighted retrieval operation.

TCABFTR, a one-byte field, contains the response code that describes the CICS/VS response to this DFHBIF TYPE=WTRETGET macro instruction. This response code can be interrogated as described under "Test Response to a Request for weighted retrieval."

Release Weighted Retrieval Storage Areas

After a series of weighted retrieval processing steps is completed, the application programmer must ensure that the VSWA established when the DFHBIF TYPE=WTRETST macro instruction is issued and the main storage

used for saving the records is released. The programmer does so by issuing a DFHBIF TYPE=WTRETREL macro instruction. Its format is:

```
┌-------------------------------------------------------------------┐
|      |        |                                                   |
|      | DFHBIF |  TYPE=WTRETREL                                    |
|      |        |[,NORESP=symbolic address]                         |
|      |        |[,INVREQ=symbolic address]                         |
|      |        |[,ILLOGIC=symbolic address]                        |
|      |        |                                                   |
└-------------------------------------------------------------------┘
```

where:

TYPE=WTRETREL
: indicates that the VSWA and other storage used by weighted retrieval are to be released.

NORESP, INVREQ, and ILLOGIC (VSAM only)
: are used to test the CICS/VS response to this request. These operands can be specified in this macro instruction or in a DFHBIF TYPE=WTRETCHK macro instruction. The meaning of each operand is discussed under "Test the Response to a Request for weighted retrieval."

Test Response to a Request for Weighted Retrieval

When the application programmer issues a request for the weighted retrieval function, he can check the response to his request to determine, in a deliberate manner, subsequent processing that should be carried out. One step in doing so is to specify the entry-point names (symbolic labels) of user-written exception-handling routines, any of which may be executed as a result of the check. This can be done in any of three ways:

1. Include the entry-point names in operands of the DFHBIF macro instruction by which the weighted retrieval function is requested.

2. Include the entry-point names in operands of a

```
DFHBIF  TYPE=WTRETCHK,                                              *
          .
          .
          .
```

   macro instruction immediately following the DFHBIF macro instruction by which the service is requested.

3. Include instructions immediately following the DFHBIF macro instruction that examine the response code set automatically by CICS/VS when making the response, and transfer control to an appropriate user-written exception-handling routine accordingly.

Under either of the first two methods above, CICS/VS checks the response code that it sets and transfers control to the exception-handling routine named in the operand associated with the condition that has occurred (if that operand has been specified). The application programmer need not be concerned with which response code corresponds to which condition. It is only necessary to understand the keyword operands and be sure that there is some provision for all conditions that may occur.

The general format of the DFHBIF TYPE=WTRETCHK macro instruction is as follows:

```
┌-------------------------------------------------------------------┐
|       |         |                                                 |
|       | DFHBIF  |  TYPE=WTRETCHK                                  |
|       |         |[,NORESP=symbolic address]                       |
|       |         |[,DSIDER=symbolic address]                       |
|       |         |[,NOTOPEN=symbolic address]                      |
|       |         |[,NOTFND=symbolic address]                       |
|       |         |[,INVREQ=symbolic address]                       |
|       |         |[,ENDFILE=symbolic address]                      |
|       |         |[,IOERROR=symbolic address]                      |
|       |         |[,OFLOW=symbolic address]                        |
|       |         |[,ILLOGIC=symbolic address]                      |
|       |         |                                                 |
└-------------------------------------------------------------------┘
```

where:

TYPE=WTRETCHK
: indicates that checking of the response to a weighted retrieval request is desired.

NORESP, DSIDER, NOTOPEN, NOTFND, INVREQ, ENDFILE, IOERROR, OFLOW, and ILLOGIC (VSAM only)
: are optional operands that test for specific CICS/VS responses to a preceding DFHBIF macro instruction. In each case, the symbolic address identifies the location to which control is transferred if the specified condition has occurred. Each keyword operand above is summarized in Figure 9-6.

When the third approach above is used, the application programmer must know the CICS/VS response codes and their meanings. (The response code is available to the application program at TCAWTRC.) The possible response codes and the conditions to which they correspond are identified in the right-hand columns of Figure 9-6. DFHBIF macro instructions for which the conditions are applicable are shown at the left.

If the application programmer does not provide for the checking for a particular response to a weighted retrieval macro instruction, and if the exception condition corresponding to that response occurs, program flow proceeds to the instruction following the weighted retrieval macro instruction in the application program.

Weighted Retrieval Request by DFHBIF Macro Instruction	Condition	Response Code	
		Assembler	ANS COBOL and PL/I
WTRETST, WTRETGET, WTRETREL, WTRETCHK	NORESP (Normal Response)	X'ØØ'	12-Ø-1-8-9
WTRETST, WTRETCHK,	DSIDER (Data Set Identification Error)	X'C1'	'A'
WTRETST, WTRETGET, WTRETCHK	NOTOPEN (Data Set Not Open)	X'C2'	'B'
WTRETST, WTRETGET, WTRETCHK	NOTFND (Record Not Found)	X'C8'	'H'
WTRETGET, WTRETCHK,	ENDFILE (End of File)	X'C4'	'D'
WTRETST, WTRETGET, WTRETREL, WTRETCHK	INVREQ (Invalid Request; see note 1)	X'C3'	'C'
WTRETST, WTRETGET, WTRETCHK	IOERROR (Input/Output Error)	X'C5'	'E'
WTRETST, WTRETGET, WTRETCHK,	OFLOW (Overflow; see note 2)	X'C6'	'F'
WTRETST, WTRETGET, WTRETREL, WTRETCHK	ILLOGIC (VSAM Error; see note 3)	X'C7'	'G'

Notes:

1. If the data set is not a VSAM file, the field TCAWRAA is set to zero. CICS/VS file control handles other errors of this type, in which case, TCAWRAA contains the address of the FCT entry for the data set.
2. For WTRETST, this response indicates that the system-defined maximum storage GETMAIN (64K) is insufficient to hold all retrieved record keys and these records have the same percentage of acceptability. In this case, the terminal operator must specify a relative record number (the relative position of the first record to be retrieved among the retrieved records) and a number of records (NRECDS) to be presented. For WTRETGET, this response means that no records were returned because all had identical percentages of match and not all could be returned because of the limit specified in NRECDS.
3. This response indicates that a VSAM error that does not fall into one of the above categories has occurred. The VSWA contains the VSAM request parameter list that contains the VSAM logical error.

Figure 9-6. Weighted Retrieval Response Codes

Example

Assume that, for purposes of state welfare applications, a VSAM file labeled SRCHFILE is maintained on magnetic disk. SRCHRECD is an area of storage that holds individual records retrieved from the file. The file is to be searched to retrieve up to 100 records that satisfy (or come closest to satisfying) the criteria: last name = SMITH, first initial = J, and mother's name = MARY.

```
            .
            .
            .
         COPY   DFHTCADS
LNAME    DS     CL18
FINIT    DS     CL1
MOM      DS     CL7
         DFHBFTCA OPTION=WTRET
            .
            .
SRCHRECD DSECT
         USING *,RCDBASE
```

```
LAST     DS      CL18
FIRST    DS      CL1
MOTHER   DS      CL7
            .
            .
         DFHBIF TYPE=WTRETST,                                          *
                DATASET=SRCHFILE,                                      *
                RDIDADR=KEYFLD,                                        *
                INPUTNO=100,                                           *
                INPUTST=10,                                            *
                INPUTPC=(100,80),                                      *
                NRECDS=50,                                             *
                NORESP=STARTOK
            .
            .
         (error processing)
            .
            .
STARTOK  DS      0H
         L       VSWABAR,TCAWRAA
         DFHBIF TYPE=WTRTPARM,                                         *
                FIELD1=(LNAME,18,C),                                   *
                FIELD2=(SRCHRECD,LAST),                                *
                MATCH=50
         DFHBIF TYPE=WTRTPARM,                                         *
                FIELD1=(FINIT,1,C),                                    *
                FIELD2=(SRCHRECD,FIRST),                               *
                MATCH=30
         DFHBIF TYPE=WTRTPARM,                                         *
                FIELD1=(MOM,7,C),                                      *
                FIELD2=(SRCHRECD,MOTHER),                              *
                MATCH=20
WRGET    DS      0H
         ST      VSWABAR,TCAWRAA
         DFHBIF TYPE=WTRETGET,                                         *
                NORESP=GETOK,                                          *
            .   ENDFILE=ENDPROC
            .
         (error processing)
            .
            .
GETOK    DS      0H
         L       RCDBASE,VSWAREA     GET ADDRESSABILITY TO RECORD
            .
            .
         (on first WTRETGET, check to see whether too many
         records have been skipped, enough records returned,
         acceptable range of % returned, and the like)
            .
            .
            .
         (process retrieved record)
         B       WRGET
            .
            .
ENDPROC  DS      0H
            .
            .
         ST      VSWABAR,TCAWRAA
         DFHBIF TYPE=WRETREL
```

CHAPTER 10. BASIC MAPPING SUPPORT

The basic mapping support (BMS) function provides the CICS/VS application programmer with many vital formatting services to assist in preparing output data streams and interpreting input data streams for the terminal network. These formatting services are provided by BMS modules that interface between the application program and the CICS/VS terminal control program.

The application program passes data to BMS and receives data from BMS in a standard non-device-dependent format. BMS macro instructions are issued by the application program to control formatting of the data and to initiate input from a terminal or output to one or multiple terminals.

BMS ADVANTAGES

Two principal advantages are realized by using the formatting services of BMS: device independence and format independence.

DEVICE INDEPENDENCE

Device independence permits the application program to provide data to a terminal or to receive data from a terminal without regard to the physical characteristics of the terminal type. Under BMS, the terminal may be any of the following devices: 1050, 2740, 2741, 2770, 2780, 3780, 2980 Models 1 and 2, 2980-4 (keyboard and printer only), 3270, TWX, tape, disk, CRLP (a device declared to have card-reader-in-line-printer-out characteristics), or TCAM terminals defined by TRMTYPE=TCAM in the DFHTCT TYPE=TERMINAL macro instruction. In addition, BMS also supports the VTAM-supported 3600 or, 3650 (host conversational and interpreter logical units only). The descriptions of the macro instructions used for BMS include VTAM-supported terminals. Refer to the CICS/VS Advanced Communication Guide, for details of BMS programming considerations unique to VTAM-supported terminal systems.

With BMS, a CICS/VS installation with multiple terminal types need only provide one program for each application transaction to support all terminal types in the installation. BMS identifies which terminal type is requesting use of the application program and provides for the conversion of the device-dependent data stream to and from the standard non-device-dependent data format used by the application program. A CICS/VS installation using a single terminal may wish to use the formatting services of BMS to facilitate the addition of other terminal types or the conversion to another terminal type in the future.

FORMAT INDEPENDENCE

Format independence permits the application program to provide data to one or more terminals or to receive data from a terminal without regard to the physical placement of fields within the data stream or on a terminal.

All references to data by the application program are through symbolic field names. The placement of fields within the data stream is accomplished by BMS through the use of information stored in data format tables called maps. A CICS/VS installation in which BMS is used may rearrange the fields to be included in a terminal message by simply

changing some values stored in the map that defines the format of the message. The application program that causes the message to be written need not be modified. The programming maintenance requirements can be considerably less than they might be if BMS were not used.

Format independence also permits certain constant information such as headings, field-identifying keywords, and 3270 screen formats to be stored in maps. These constants can be modified simply by changing their values in the maps. Any programs that refer to the maps benefit from the changes, but none of the programs themselves need be modified.

The format independence provided by BMS may be compared with the independence provided by DL/I for data bases. Both remove from the application program the requirement to know the physical placement of fields within the data record or message. Fields may be physically rearranged, removed, or added without necessitating program maintenance on all application programs using the record or message.

BMS TECHNIQUES

The primary techniques by which BMS provides formatting services are: data mapping and formatting, terminal paging, and message routing.

DATA MAPPING AND FORMATTING

Data mapping is the technique used by BMS to convert the standard non-device-dependent data format, which the application program uses, to and from the device-dependent data stream required for the particular terminal type in use. Device-dependent control characters are embedded or removed by BMS during this processing.

The application program may select either of three standard data formats in which to provide or accept data from BMS: field data format, block data format, and text data format.

When field data format is used, data is provided to BMS as separate fields. Each field is given a symbolic field name by the application programmer. This name is used when passing data to, or retrieving data from, BMS. Each field consists of a two-byte length (used by BMS on input), a one-byte attribute (for 3270 output operations), and the field data area. A map describing the field position, data length, and other information about each field is created to control the mapping function.

When block data format is used, data is provided to BMS as line segments. Fields positioned within the line segments may be given symbolic field names to aid the application program in positioning the fields. Each field provides for a one-byte attribute and the field data area. A gap consisting of several blanks may separate consecutive fields in the line segment. A map is used to describe the number and lengths of line segments, the field positions, data lengths, and other necessary information.

When text data format is used, output data consisting of a data stream with optional new-line (X'15') characters is provided to BMS. Device independence divides the data stream into lines no longer than those defined for the particular terminal to which the data stream is related. Character strings (non-blanks, X'40) or words which overlap lines are placed unbroken on the next available line. If new-line characters are included in the data stream to further define line lengths, they are honored. CICS/VS inserts the appropriate leading characters, carrier returns, and idle characters, and eliminates

trailing blanks from each line. If tab control characters are contained in the data stream, the user should also supply

all of the necessary new-line characters. No maps are used with text data format.

Field data format is the most common standard data format for both video and hardcopy terminals. Block data format may be used with both video and hardcopy terminals, but it is more useful for input operations on hardcopy terminals. Text data format is used with both video and hardcopy terminals and is especially convenient when programming to handle data not divided into fields. When text data format is used with a 3270 device, an attribute byte appears on the 3270 as a blank at the beginning of each line and in front of each new piece of data.

TERMINAL PAGING

Terminal paging permits the application program to (1) combine several small mapped data areas into one or more pages of output, or (2) prepare more output than can be contained in one page of output. By definition, a page is the physical area of a terminal on which data can be displayed or printed at one time. The size of the area (in numbers of lines and columns) is specified for the particular terminal in the CICS/VS terminal control table by the system programmer.

Since a page of output may be constructed by BMS from several small maps, it is convenient to generate these maps together in a map set. By definition, the map set is a collection of maps generated and stored together in the CICS/VS program library. A reference to one map in the map set causes the entire map set to be loaded into storage for the duration of the task or until another map set is referred to by the task. DFHMSD, DFHMDI, and DFHMDF macro instructions, described later, in this chapter are used in constructing the map set.

At execution time, the application program issues DFHBMS TYPE=PAGEBLD macro instructions to map and position portions of an output page. If all data to be mapped cannot be contained on one page, BMS recognizes the overflow condition and can transfer control to an overflow routine within the application program. This routine normally causes the current page to be written tc temporary storage, a new page to be started, a heading to be placed on the new page, and the data causing the overflow to be mapped on the new page. As each page of the output message is completed, the page is written to temporary storage to await completion of the multiple pages of the output. The result of one or more BMS requests for output services, all of which have the same disposition (OUT, STORE, or RETURN, as explained later in this chapter), is known as a logical message. To cause the logical message to be completed, the application program issues a DFHBMS TYPE=PAGEOUT macro instruction. Alternatively, the logical message is completed upon termination of the application program unless a short on storage condition exists, in which case the logical message is purged.

Terminal paging provides the additional function of building a logical message without the use of maps. A DFHBMS TYPE=TEXTBLD macro instruction is issued to request this type of page building. The data is provided to BMS as text data, which BMS places on succeeding lines (and pages, if necessary) without reference to maps. A word is not split between lines; any word that cannot fit on the remaining portion of a line is placed on the next line. The formatting of the logical message can be controlled through the data itself by embedding new-line characters (X'15') within the text data. To cause the TEXTBLD logical message to be completed, the application program issues a DFHBMS TYPE=PAGEOUT macro instruction or terminates execution.

DFHBMS TYPE=TEXTBLD and TYPE=PAGEBLD macro instructions cannot be used to build portions of the same logical message. The process of building a logical message can be discontinued by means of a DFHBMS

TYPE=PURGE macro instruction. This instruction purges the portions of the message already built in main storage or on temporary storage.

MESSAGE ROUTING

Message routing permits an application program to concurrently build and route a logical message to one terminal or to multiple terminals. The message is automatically scheduled for each designated terminal, to be delivered as soon as the terminal is available to receive messages or at some future time.

The terminal paging facility of BMS is used for message routing so the design of application programs is very similar for the two facilities. In fact, Message Routing may be considered as a "one macro instruction enhancement" to terminal paging.

To initiate a routing operation, the application program issues a DFHBMS TYPE=ROUTE macro instruction followed by DFHBMS TYPE=PAGEBLD or TYPE=TEXTBLD instructions to build the logical message to be routed. A DFHBMS TYPE=PAGEOUT macro instruction terminates the page building and causes the message to be routed. A routed logical message may be one or more pages in length.

A parameter of the DFHBMS TYPE=ROUTE macro instruction points to a list of terminals to receive the routed logical message. The list may contain the terminal identification and operator identification of each terminal designated to receive the message. If only a terminal identification is specified, the message is routed to that terminal, regardless of who is signed on at the terminal. If both the terminal identification and the operator identification are specified, the message is routed to the terminal but delivered only when the specified operator is signed on. If only the operator identification is specified, BMS scans the terminal control table and delivers the message to the first terminal at which the operator is signed on.

Another parameter of the DFHBMS TYPE=ROUTE macro instruction is a specific operator class code. If specified, only an operator signed on with that class code may receive the routed message. One to twenty-four class codes may be assigned to operators in the CICS/VS sign-on table.

The DFHBMS TYPE=ROUTE macro instruction further designates whether the logical message is to be delivered as soon as possible or at a specific time or some interval of time in the future. If the routed logical message cannot be delivered within a specified length of time, an error message may be returned to the terminal sending the message or to some designated alternative terminal. The logical message may be purged, or it may be retained indefinitely -- until delivered or until deliberately purged by an operator at the receiving terminal.

If a logical message is to be routed to more than one terminal type, BMS builds the message for each terminal type. Each message is stored on temporary storage until all terminals of the related terminal type have received the message. If a terminal is scheduled to receive a message but is not eligible, the message is preserved until one of the following conditions occurs:

- The terminal operator requests display of the message.
- A change in terminal status allows the message to be sent.

- A time period (specified at system generation) has elapsed, causing the message to be purged.
- The message is purged through the destination terminal.

Since maps unique to a terminal type may be used with message routing, the arrangement of fields and headings or constants may differ from one terminal type to another. This permits some tailoring of the various messages, although a single application program is used to build and route the messages.

Another consideration of routing to different terminal types is the handling of overflow conditions. Since different terminal types may have different page sizes, the overflow condition is apt to occur at different times in page building. BMS not only can return control to an overflow routine in the application program, but also can indicate which terminal type caused the overflow and how many pages have been created for that terminal type.

The message routing facility of BMS is an ideal tool for developing message switching and broadcasting applications. CICS/VS provides a generalized message switching application program that uses the message routing facility of BMS (see the CICS/VS System Administrator's Guide).

PROGRAMMING CONSIDERATIONS

As a preparatory step to using the BMS mapping function in a particular application, two types of maps are assembled offline through use of CICS/VS macro instructions: (1) a physical map used by CICS/VS to convert native-mode data into the format desired by the application programmer, and (2) a symbolic description map used by the application programmer to symbolically refer to the data in the terminal buffer. The physical map is a table of information about each field which is stored in the CICS/VS program library to be loaded by BMS at execution time. The symbolic description map is a set of source statements which are cataloged into the appropriate source library (Assembler, American National Standard (ANS) COBOL, or PL/I) and copied into the application program when it is assembled or compiled.

The user defines and provides names for fields and groups of fields that may be written to and received from the devices supported by BMS. The symbolic description map can be copied into each application program that uses the associated physical map. Data is passed to and from the application program under the field names in the symbolic description map. Since the application program is written to manipulate the data under the field names, altering the map format by adding new fields or rearranging old fields does not necessarily alter the program logic.

If the map format is altered, it is necessary in most cases to make the appropriate changes to the macro instructions that describe the map and then reassemble both the physical map and the symbolic description map. The new symbolic description map must then be copied into the application program and the program reassembled. There are certain map alterations that can be made without necessitating reassembly of the symbolic description map.

An application program has access to the input and output data fields using the names supplied to the fields when the maps were generated. The application logic should be dependent upon the named fields and their contents but should be independent of the relative positions of the data fields within the terminal format. If it becomes necessary to reorganize or add to a map format, the existing application program must be reassembled to gain access to the new positions of these data

fields. Reprogramming is not necessary to account for new fields or for the changed terminal format of those fields.

By using BMS to construct and interpret data streams, application programmers can insulate application programs from the device-dependent considerations required to handle the data streams. If necessary, the application program has the facility to temporarily modify the attributes or the initial data of any named field in an output map. A collection of named attribute combinations is supplied within BMS so that the application program remains essentially independent of the data stream format.

The ability to progressively add to map definitions without obsoleting existing application programs permits the design and implementation of systems in a modular fashion with a progressive expansion of the screen formats. Design and programming of the first stages of applications can begin before later stages have been designed. These early implementations are protected from updates in the terminal formats.

Note: To use pre-VS applications requiring the BMS mapping function on terminals other than the 3270 Information Display System, the application programs, physical maps, and symbolic description maps must be reassembled.

SPECIFYING MAPS FOR BASIC MAPPING SUPPORT

A map to be used for input or output operations must be specified for BMS. After the user has defined all maps related to a particular application, he can group the maps into a map set. This grouping technique permits the formatting of a page of output consisting of one or more of the maps in the map set. If the map set has been placed in the CICS/VS program library, the user must use the MAPSET=map set name and MAP=map name specifications in any DFHBMS macro instruction requesting an operation in which the map is required. If preferred, the user may place the seven-character name of the map set at TCAMSMSN and specify MAPSET=YES and place the name of the map at TCABMSMN and specify MAP=YES in the DFHBMS macro instruction.

An Assembler-language programmer may "hard code" a map set in a program, place the address of the map set at TCAMSMSA, and code MSETADR=YES. If desired, the user may code MSETADR=symbolic address, where symbolic address is the label of the hard-coded map set. The MAP=map name specification must also be provided with the MSETADR parameter to locate a specific map within the hard-coded map set.

Map sets placed in the CICS/VS program library are accessed by BMS through program control DFHPC TYPE=LOAD macro instructions. Therefore, each map set name must be entered in the processing program table (PPT) by the system programmer. Any device-dependent map sets must be placed in the CICS/VS program library. They must be identified by the device-dependent suffixed name (see programming note 1 under "DFHMSD Macro Instruction"), and a corresponding entry of the same name must appear in the PPT. The application programmer should realize that when a map set is specified, BMS scans the PPT at least once, and perhaps twice, for each device-dependent map set that is to be used by the program.

IMPLIED READ/WRITE

Input and output requests result in a terminal control READ and WRITE, respectively. Therefore, the user is not required to code any terminal control (DFHTC) macro instructions.

Nothing prevents the user from alternately coding native-mode and BMS operations. A MAP-only operation can be requested to cause BMS to map a native-mode input TIOA. If a MAP operation is requested for input from a nonformatted 3270 buffer, mapping is not performed and a nonformatted native-mode TIOA is returned to the application program.

Note: The read that contains the transaction code and causes initiation of the transaction is a native-mode data stream. The MAP request may be used to convert this TIOA to a mapped TIOA.

MAP DEFINITION

Input Mapping

Input maps are defined using the DFHMSD, DFHMDI, and DFHMDF macro instructions during offline map generation. All maps related to a particular application are grouped to form a map set (see "Offline Map Building").

The maximum data length and the starting position of each field to be read must be defined. This operation produces a map and a symbolic storage definition of data placed into a TIOA by BMS during execution of a CICS/VS application program. The physical map is used by BMS to construct the TIOA as defined by the symbolic storage definition. The data returned to an application program from an input mapping operation is in TIOA format.

The TIOA symbolic storage definition contains an area for the length of each input data field, followed by a flag byte and an area for the data read. Space is reserved for the maximum number of bytes defined for each field.

The length specified for a field may differ from the number of characters that are entered for the field at program execution time. (The maximum length allowed for a field is 256 characters.) If more data is keyed than specified in the map, the data is truncated on the right to the number of characters specified. The length that is returned to the application program is the truncated length. If less data is keyed than specified, the remaining character positions are filled with blanks or zeros and the length of the keyed data is returned in the length field.

Any fields that are entered as input but are not defined by the symbolic description map are discarded. The length and data areas of any field defined but not keyed are set to nulls (X'00'). An X'80' is placed in the first byte (attribute byte) ahead of the data area of any field that has been modified to all nulls (for example, by an erase EOF key). The length area is set to zeros.

One byte is reserved for a pen-detectable field. This byte contains a hexadecimal 'FF' if the field is selected or a hexadecimal '00' if the field is not selected. The length area of a pen-detectable field contains a binary one if selected or a binary zero if not selected.

The program can access the length, flag, and data of any field by symbolic labels. The length area is a halfword binary field and is addressed by the name "fieldname.L" or "groupname.L". The flag is a one-byte field and is addressed by the name "fieldname.F" or "groupname.F". The data portion of each field (or group of fields) is contiguous with the length and flag areas. A group of fields, or a single field not within any group of fields, has one data portion addressed by the name "groupname.I" or "fieldname.I". For fields contained within a group, there are no intervening length or flag areas

(only "groupname.L" exists) but each field can be addressed by a name "fieldname.I".

Note that "." is a concatenation symbol used here for documentational purposes; it should not be used when referring to either the data or the data length. For example, in the case of field name XYZ, the data is referenced as XYZI; the data length is referenced as XYZL; and the flag is referenced as XYZF.

Output Mapping

Output maps, like input maps, are created offline during map generation using the DFHMSD, DFHMDI, and DFHMDF macro instructions (see "Offline Map Building"). Each field to be displayed must be defined as to starting location, length, field characteristics, and default data (if desired).

When defining fields, the user may provide a name for any field that he desires to refer to at execution time. Such names are associated with the fields in the symbolic storage definition of the TIOA to allow symbolic references to be made to them. The user may specify not only the characteristics of the field but also the default data to be written as output for a field when no data is supplied for that field by an application program. This facility permits the specification of titles, headers, and so forth, for output maps. The user may temporarily override the field characteristics, the data, or both field characteristics and data of any field for which he has specified a name. The desired changes are simply inserted into the TIOA under the has been specified field name in the symbolic storage definition (symbolic description map) in the program.

Note: Output field data, whether initial map data or data supplied by the program, must not begin with a null character (X'00'). In order for displayable data to be mapped into the output data stream, blank characters (X'40') must be used to position the data in the field.

The fields of an output map are assigned names as specified in the DFHMDF macro instruction. The characteristic or attribute byte is named "fieldname.A" or "groupname.A". For a field contained within a group, the data area is given the name "fieldname.O", but there is no separate attribute byte for the field. (Only the group name has the attribute byte.) For a group name, or a field not contained within a group, the data area is given the name "groupname.O" or "fieldname.O." A field not contained within a group is treated as a group containing one field entry. An unused two-byte length field precedes each attribute byte and data field to provide a format similar to an input symbolic storage description TIOA. Application programs written to use pre-VS data formats are source compatible if all references to TIOA data are symbolic.

Note that "." is a concatenation symbol used here for documentational purposes; it should not be used when referring to either the data or the data attributes. For example, in the case of field name XYZ, the data is referred to as XYZO; the attribute byte is referred to as XYZA.

Pen-detectable fields should be "auto skip" to prevent data from being keyed into them. Because of the nature of pen-detectable fields, in most instances, they should not be modified. If the data field is modified, the application program must ensure that the first character is a "?", ">", or blank character; otherwise, the field is no longer pen-detectable.

Fields that can be keyed should be delineated by a stopper field to ensure that all the data keyed and transmitted can be mapped.

Input/Output Mapping

Input/output (INOUT) maps combining all the functions of input and output maps can be created offline during map generation using the DFHMSD, DFHMDI, and DFHMDF macro instructions, as explained below.

OFFLINE MAP BUILDING

Offline map building is accomplished through use of three types of macro instructions: DFHMSD macro instruction, which defines a map set; DFHMDI macro instruction, which defines a map within a map set; DFHMDF macro instruction, which defines a field within a map.

The general formats of these macro instructions are shown below; representative coding and symbolic storage definitions generated from these are shown in Figures 10-8 through 10-17 at the end of this chapter.

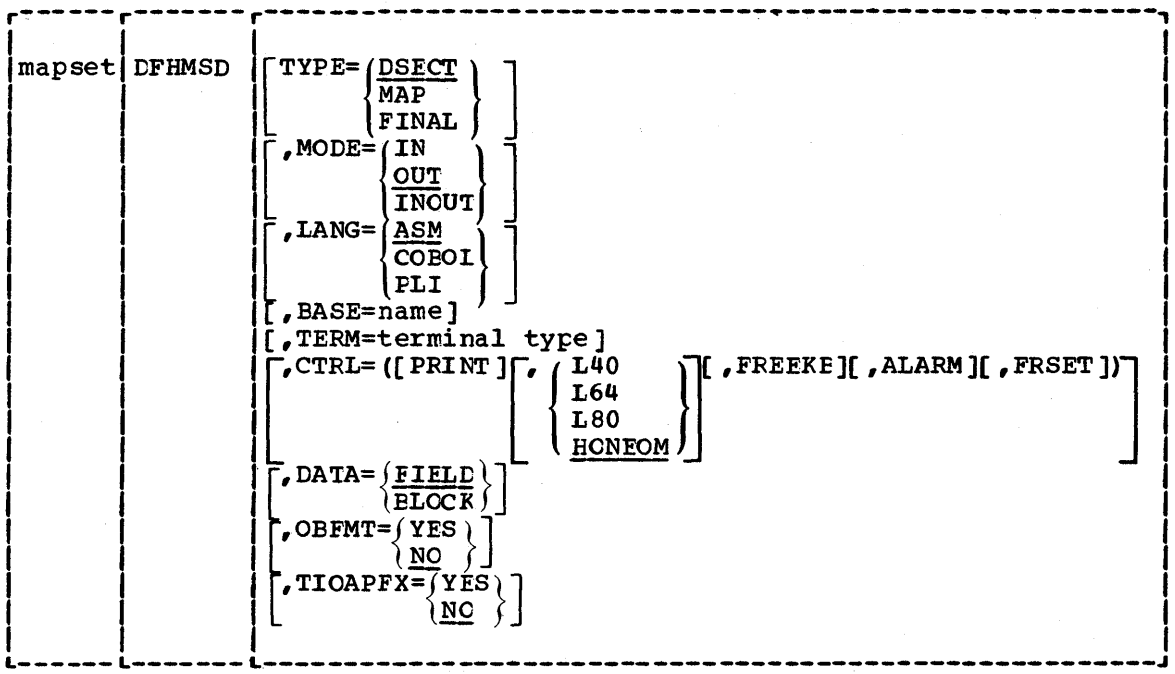
Note: Pre-VS versions of CICS BMS provided offline map generation support for the 3270 Information Display System through DFHMDI and DFHMDF macro instructions. For compatibility, DFHMDI and DFHMDF macro instructions written to use this support will be assembled correctly under CICS/VS BMS.

DFHMSD Macro Instruction

Because a page of output can be constructed from several maps, it is convenient to group these maps into a map set. BMS generates and stores map sets in the CICS/VS program library under the names selected by the application programmers. A reference to one map in the map set causes the entire map set to be loaded into storage for the duration of the task, or until another map set is referred to by the task.

Information pertaining to an entire map set is specified in the DFHMSD macro instructions which always appears at the beginning and end of each map set generation. The one at the beginning indicates whether physical maps or symbolic description maps are being generated; the one at the end indicates the end of the map set generation.

All operands other than the TYPE operand of a DFHMSD macro instruction are the same for a physical map generation run and for the corresponding symbolic description map generation run. The application programmer should specify TYPE=MAP for the former, and TYPE=DSECT for the latter. The format of the DFHMSD macro instruction is as follows:



where:

mapset
: is the one- to seven-character name of this map set, to be specified in the MAPSET operand of any DFHBMS macro instruction that refers to this map set. The name must begin with an alphabetic character and, if the map is to reside in the CICS/VS program library, must differ from other map names or program names (see programming note 1, following).

TYPE=
: indicates the generation function of the macro instruction.

DSECT
: indicates that this is a symbolic description map generation run to create the list of field names tc be copied into an application program. If a single map set is to be used by application programs written in different languages, a separate DFHMSD TYPE=DSECT macro instruction must be written for each language to put the table of field names into the copy library of the language.

MAP
: indicates that this is a physical map generation run to create the control information block used by BMS to perform mapping. This physical map is stored in the CICS/VS real-time relocatable program library and loaded as required by BMS. The Assembler-language application programmer can generate the map in his program and pass the address of the map to BMS instead of using this facility to generate and store the map beforehand in the CICS/VS program library.

FINAL
: must be coded as part of the last macro instruction of a map definition. If other parameters are coded in the DFHMSD TYPE=FINAL macro instruction, they are ignored.

MODE=

IN
indicates an input map generation.

OUT
indicates an output map generation.

INOUT
indicates that the map definition is to be used for both input and output mapping operations.

Note: Input mapping is not available for VTAM-supported 3600 terminals. However, INOUT may be specified for map generation. The map can then be used as a dummy input map for input operations using the DFHBMS TYPE=IN macro instruction.

LANG=

specifies the language in which the application program referring to a symbolic description map is written and, hence, is applicable for only a DFHMSD TYPE=DSECT macro instruction.

ASM
indicates that the symbolic description map is to be referred to by an Assembler-language program.

COBOL
indicates that the symbolic description map is to be referred to by an ANS COBOL program.

PLI
indicates that the symbolic description map is to be referred to by a PL/I program.

BASE=name

is used to group symbolic description maps under one name, specified in this DFHMSD TYPE=DSECT macro instruction and in a similar macro instruction for any other symbolic description map to be included in the group. This operand is valid for symbolic description maps generated for use in an ANS COBOL or a PL/I program; it is not applicable for Assembler-language programs (see programming note 3).

TERM=terminal type

where terminal type is one of the following:

1050, 2740, 2741, 2770, 2780, 3780, 2980-4, 2980, TAPE, DISK, TWX, CRLP, 3270-1, 3270-2, 3270, 3601, 3653, 3650UP, 3650/3270, ALL

indicates the type of output device related to this map set. Use CRLP or ALL for TCAM terminals. A suffix is appended to the map set name to indicate the terminal type specified (see programming note 1). If no terminal type is specified, 3270 is assumed. (See programming note 2.)

The 3650 TERM specifications are defined as follows: 3653 for the host-conversational (3653) logical unit, 3650UP for the interpreter logical unit, and 3653/3270 for the host-conversational (3270) logical unit.

CTRL=

is used to specify device characteristics related to terminals of the 3270 Information Display System. CTRL=ALARM is valid for VTAM-supported terminals; all other parameters for CTRL= are ignored.

PRINT
must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed. This operand is ignored if the map set is used with 3275s without the Printer Adapter feature or with 3277s.

L40, L64, L80, HONEOM
are mutually exclusive options that control the line length on the printer. L40, L64, and L80 force a carrier return/line feed after 40, 64, or 80 characters, respectively. HONEOM causes the printer to honor all new-line (NL) characters and the first end-of-message (EM) character that appear in displayable fields of the data stream. If the latter option is specified, the application program must insert the NL and EM characters into the data stream. If the NI character is omitted, a carrier return/line feed occurs at the physical end of the carriage. If the EM character is omitted, printing stops at the end of the 3270 buffer.

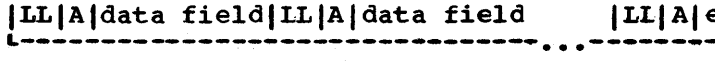
FREEKB
specifies that the keyboard should be unlocked after this map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM
activates the 3270 audible alarm feature. For a VTAM terminal ALARM signals BMS to set the alarm flag in the function management header.

FRSET
indicates that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before any map data is written to the buffer. This allows the DFHMDF ATTRB specification for the requested map to control the final status of any fields written or rewritten in response to a DFHBMS macro instruction.

DATA=
specifies the format of the data as seen by the application program.

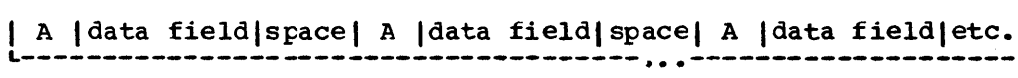
FIELD
indicates that the data is passed as contiguous fields in the following format:



LL is two bytes specifying the length of the data as input from the terminal (this field is ignored in output processing). A is a byte into which the programmer may place an attribute to override that specified in the map used to process this data (see "Standard Attribute List and Printer Control Characters (DFHBMSCA)").

BLOCK
indicates that the data is passed as a continuous stream which is processed as line segments of the length specified in the map used to process this data set. The data is in the form that it appears on the terminal; that is, it contains data fields and interspersed blanks corresponding to any spaces that are to appear between the fields on

output. The first byte of each line is the attribute byte; it is not available for data.



See programming note 4.

OBFMT=
specifies whether outboard formatting is to be used. This operand is available only for 3650 logical units. Refer to the CICS/VS Advanced Communication Guide for details on 3650 logical units and on outboard formatting.

YES
indicates that all maps within this mapset are eligible for use in outboard formatting, except those for which OBFMT=NO is specified in the DFHMDI macro instruction.

NO
indicates that no maps within this mapset are eligible for use in outboard formatting, except those for which OBFMT=YES is specified in the DFHMDI macro instruction.

TIOAPFX=
specifies whether BMS should include a filler in the symbolic TIOA description(s) to allow for the unused TIOA prefix. If this operand is coded, the same storage address may be used for TIOABAR and the map base.

YES
indicates that the filler should be included in the symbolic TIOA description(s). This operand is ignored unless TYPE=DSECT and LANG=COBOL or PLI is coded. If TIOAPFX=YES is coded, all maps within the mapset have the filler, except when TIOAPFX=NO is coded on the DFHMDI macro instruction.

NO
is the default and indicates that the filler is not to be included. The filler may still be included for a specific map if TIOAPFX=YES is coded on the DFHMDI macro instruction.

Programming Notes:

1. A suffix based on the terminal type specified in the TERM operand of a DFHMSD macro instruction is appended to the map set name at assembly time. When a mapping operation is requested by means of a DFHBMS macro instruction in an application program, CICS/VS automatically appends a similar suffix to the map set name specified in that instruction. This enables CICS/VS to locate a device-dependent map set for a requested operation but maintains device independence in the application program. The suffix appended by CICS/VS is one of the following:

Terminal Type	Map Set Suffix
CRLP	A
TAPE	B
DISK	C
TWX	D
1050	E
2740	F
2741	G

2770	I
2780	J
3780	K
3270-1	L
3270-2	M
2980 (except Model 4)	Q
2980-4	R
3270	blank
3601	U
3653	V
3650UP	W
3650/3270	X
ALL (of the above)	blank

The map set suffix for CRLP or ALL (whichever was defined by the TERM operand in the DFHMSD macro instruction) is appended for TCAM terminals.

2. The application programmer who specifies ALL in the TERM operand must be certain that device-dependent characters are not included in the map set and must ensure that format characteristics such as page size are suitable for all input/output operations (and all terminals) in which the map set will be applied. For example, the 3270-1 terminal is limited to 480 bytes; the 3270-2 terminal is limited tc 1920 bytes; the 3604 is limited to six lines of 40 characters each. Within these guidelines, use of ALL can offer important advantages. Since an assembly run is required for each map generation, a specification of ALL, indicating that one map is to be used for multiple terminals, can result in significant time and storage savings.

3. The BASE operand of a DFHMSD macro instruction is applicable only for symbolic description maps to be referred to by macro instructions in an ANS COBOL or a PL/I application program. As an example, assume that the following DFHMSD macro instructions are used to generate symbolic description maps (symbolic storage definitions) for two map sets.

```
MAP1    DFHMSD TYPE=DSECT,                                    *
               TERM=2780,                                     *
               LANG=COBOL,                                    *
               BASE=DATAREA1,                                 *
               MODE=IN

MAP2    DFHMSD TYPE=DSECT,                                    *
               TERM=3270,                                     *
               LANG=COBOL,                                    *
               BASE=DATAREA1,                                 *
               MODE=OUT
```

The symbolic storage definitions of this example might be referred to in an ANS COBOL application program as follows:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   .
   .
   02 TIOABAR PICTURE S9(8) COMPUTATIONAL.
   02 MAPBASE1 PICTURE S9(8) COMPUTATIONAL.
   .
   .

01 DFHTIOA COPY DFHTICA.
01 DATAREA1 PICTURE X(1920).
01 name COPY MAP1.
```

01 name COPY MAP2.

MAP1 and MAP2 multiply redefine DATAREA1; only one 02 statement is needed to establish addressability. However, the program can only use the fields in one of the symbolic map areas at a time.

If BASE=DATAREA1 is deleted from this example, an additional 02 statement is needed to establish addressability for MAP2; the 01 DATAREA1 statement is not needed. The program can refer to fields concurrently in both symbolic map areas.

In PL/I application programs, the name specified in the BASE operand is used as the name of the pointer variable on which the symbolic storage definition is based. If this operand is omitted, the default name (EMSMAPBR) is used for the pointer variable. The PL/I programmer is responsible for establishing addressability for the based structures.

4. The data type associated with any map is affected by the DATA specifications, or lack thereof, in both the DFHMSD and DFHMDI macro instructions. If a specification is made in:

   - DFHMSD but not in DFHMDI, then the specification in the DFHMSD holds.
   - DFHMDI whether or not DFHMSD is specified or whether or not there is a DATA specification in the DFHMSD, then the specification in DFHMDI holds.
   - Neither DFHMSD nor DFHMDI, then DATA=FIELD is the default.

DFHMDI Macro Instruction

The DFHMDI macro instruction is used to define a specific map. It tells the size of the data to be mapped and its position within the input or output. When defining multiple maps within a map set, multiple DFHMDI macro instructions must be used. If the maps are for use in an ANS COBOL program, they must be specified in descending size sequence (the largest map first, the smallest map last). The format of the DFHMDI macro instruction is as follows:

```
map   DFHMDI  [SIZE=(line,column)]
              [,LINE={number}]
                     {NEXT  }
                     {SAME  }
              [,COLUMN={number}]
                       {NEXT  }
                       {SAME  }
              [,JUSTIFY=([{LEFT }][,{FIRST}])]
                         [{RIGHT}]  [{LAST }]
              [,HEADER=YES]
              [,TRAILER=YES]
              [,DATA={FIELD}]
                     {BLOCK }
              [,OBFMT={YES}]
                      {NO }
              [,TIOAPFX={YES}]
                        {NO }
```

where:

map
: is the one- to seven-character name of this map, to be specified in the MAP operand of any DFHBMS macro instruction that refers to this map.

SIZE=
: gives the dimensions of a map in terms of length and width.

  line
  : is a value from 1 to 240, indicating the length of a map as a number of lines.

  column
  : is a value from 1 to 240, indicating the width of a map as a number of characters per line. Space for the attribute byte should be included in the column specification.

  The SIZE operand is required in the following cases:

  - A POS=(line,column) specification is given in a DFHMDF macro instruction defining a specific field within this map.
  - This map is to be referred to in a DFHBMS TYPE=PAGEBLD macro instruction.
  - This map is to be used when referring tc input data from other than a 3270 terminal in a DFHBMS TYPE=IN or DFHBMS TYPE=MAP macro instruction.

LINE=
: specifies the starting line on a page in which data for a map is to be formatted.

  number
  : is a value from 1 to 240, indicating a starting line number. A request to map data on a line and column that has been formatted in response to a preceding request causes the current page to be treated as though complete. The new data is formatted at the requested line and column on a new page.

NEXT
indicates that formatting of data is to begin on the next available completely empty line.

SAME
indicates that formatting of data is to begin on the same line as that used for a preceding DFHBMS request. If the data does not fit on the same line, it is placed on the next available completely-empty line.

See programming note 1 for a summary of LINE and JUSTIFY relationships.

COLUMN=
specifies the column in a line at which the map is to be placed, that is, it establishes the left or right map margin. The JUSTIFY specification controls whether map and page margin selection and column counting are to be done with reference to the left or right side of the page. The columns between the specified map margin and the page margin are not available for subsequent use on the page for any lines included in the map.

number
is the column from the left or right page margin where the left or right map margin is to be established.

NEXT
indicates that the left or right map margin is to be placed in the next available column from the left or right on the current line.

SAME
indicates that the left or right map margin is to be established in the same column as the last map used that specified COLUMN=number and the same JUSTIFY parameters as this macro instruction.

See programming note 1 for a summary of COLUMN and JUSTIFY relationships.

JUSTIFY=
describes the margins on a page in which a map is to be formatted.

LEFT
indicates that the map is to be positioned starting at the specified column from the left margin on the specified line.

RIGHT
indicates that the map is to be positioned starting at the specified column from the right margin on the specified line.

FIRST
indicates that the map is to be positioned as the first map on a new page. Any partially formatted page from preceding DFHBMS requests is considered to be complete. This operand can be specified for only one map per page.

LAST
indicates that the map is to be positioned at the bottom of the current page. This operand can be specified for multiple maps to be placed on one page. However, maps other than the first map for which it is specified must be able to be

positioned horizontally without requiring that more lines be used.

LEFT and RIGHT are mutually exclusive, as are FIRST and LAST. If neither LEFT nor RIGHT is specified, LEFT is assumed. If neither FIRST nor LAST is specified, the data is mapped at the next avaliable position as determined by other parameters of the map definition and the current mapping operation. FIRST and LAST are ignored unless PAGEBLD is specified, since otherwise only one map is placed on each page.

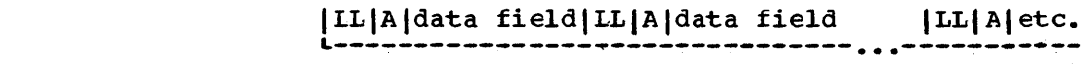
See programming note 1 for a summary of JUSTIFY, LINE, and COLUMN relationships in controlling map positioning.

HEADER=YES
allows this map to be used during PAGEBLD overflow without terminating the overflow condition (see "PAGEBLD Overflow Processing" in this chapter). This operand may be specified for more than one map in a map set.

TRAILER=YES
allows this map to be used during PAGEBLD overflow without terminating the overflow condition (see "PAGEBLD Overflow Processing"). This operand may be specified for more than one map in a map set. If a trailer map is used other than in the overflow environment, the space normally reserved for overflow trailer maps is not reserved while mapping the trailer map.

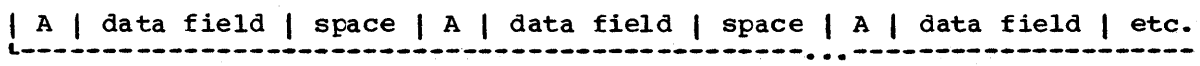
DATA=
specifies the format of the data as seen by the application program.

FIELD
indicates that the data is passed as contiguous fields in the following format:



LL is two bytes specifying the length of the data as input from the terminal (this field is ignored in output processing). A is a byte into which the programmer may place an attribute to override that specified in the map used to process this data. (See "Standard Attribute List and Printer Control Characters (DFHBMSCA).")

BLOCK
indicates that the data is passed as a continuous stream which is processed as line segments of the length specified in the map used to process this data set. The data is in the form that it appears on the terminal; that is, it contains data fields and interspersed blanks corresponding to any spaces that are to appear between the fields on output. The first byte of each line is the attribute byte; it is not available for data.



A DATA specification in a DFHMDI macro instruction overrides

any DATA specification in a DFHMSD macro instruction setting up characteristics for maps within a map set.

OBFMT=
specifies whether outboard formatting is to be used. This operand is available only for 3650 logical units. Refer to the CICS/VS Advanced Communication Guild for details on 3650 logical units and on outboard formatting.

If OBFMT is not coded in the DFHMDI macro instruction, the OBFMT specification in the DFHMSD macro instruction is used.

YES
indicates that this map is to be used with outboard formatting.

NO
indicates that this map is not to be used with outboard formatting.

NO
indicates that this map is not to be used with outboard formatting.

TIOAPFX=
specifies whether or not BMS should include a filler in the symbolic TIOA description tc allow for the unused TIOA prefix. If this operand is coded, the same storage address may be used for TIOABAR and the map base. If this operand is not coded, the TIOAPFX specification derived from the DFHMSD macro is used. There is no default for this operand on the DFHMDI macro.

YES
indicates that the filler should be included in the symbolic TIOA description for this map. This operand is ignored unless TYPE=DSECT and LANG=COBOL or PLI is coded on the DFHMSD macro instruction.

NO
indicates the filler is not to be included for this map.

Programming Notes:

1. Map positioning on a page is controlled jointly by the LINE, COLUMN, and JUSTIFY specifications. The effects of these specifications, taken together, can be summarized as follows:

Line Position	LINE=1-240	LINE=SAME	LINE=NEXT
JUSTIFY=FIRST	Line number on new page	Top of new page	Top of new page
JUSTIFY=LAST	Bottom of page	Bottom of page	Bottom of page
Neither	Line number	Same line	Next full line

Column Position	COLUMN=1-240	COLUMN=SAME	COLUMN=NEXT
JUSTIFY=LEFT	Columns from left	Same column from left	Next column from left
JUSTIFY=RIGHT	Columns from right	Same column from right	Next column from right

Page Overflow	
	JUSTIFY=FIRST LINE=nn,COLUMN=nn in formatted area Map won't fit on remainder of page

2. If the line and column specifications in the DFHMDI macro instruction cannot be satisfied during a particular operation, default action is as follows:

Line / Column	1-240	SAME or NEXT
1-240, SAME, or NEXT	Current page is completed; data is mapped on the next page at the line and column specified.	Data is mapped on the next available full line at the column specified.

DFHMDF Macro Instruction

The DFHMDF macro instruction is used to define a specific field of a map. One DFHMDF macro instruction is required for each field, giving information such as symbolic field name, field position, field length, attribute byte (for 3270 terminals), initial constant data, justification of input, and ANS COBOL or PL/I data picture. The format of this macro instruction is as follows:

```
[fld]  DFHMDF  [POS={number      }]
               [   {(line,column)}]
               [,ATTRB=([{ASKIP }][,NUM][,{BRT }][,DET][,IC][,FSET])]
               [         {PROT  }       {NORM}]
               [         {UNPROT}       {DRK }]
               [,LENGTH=number]
               [,JUSTIFY=([{LEFT }][,{BLANK}])]
               [         {RIGHT}]  [{ZERO }]
               [,INITIAL='any user information']
               [,GRPNAME=group name]
               [,OCCURS=number]
               [,PICIN='value']
               [,PICOUT='value']
```

where:

fld
: is the one- to seven-character name of this field used as a symbolic reference to the specific area of a map by the application program (see programming note 1 below).

  Note: Although specification of a field name is not required for a field within a map, a field name must be specified for at least one field of any map to be compiled under ANS COBOL or PL/I.

POS=
: is used to specify the individually addressable character location in a map at which the attribute byte that precedes this field is positioned (see programming note 2). Specification of the DFHMDF macro instruction must be sequenced by the POS operand except for output mapping operations using DATA=FIELD.

  For all fields, other than the first, within the same group name, the POS operand must specify the last position of the previous field.

  When a position number is coded which represents the last character position (479 or 1919) in the 3270, then two special rules apply:

  - The IC attribute should not be coded on that DFHMDF macro. The cursor may be set to location zero by using the cursor operand of the DFHBMS macro.
  - If the field is to be used in an output mapping operation with the data=only specification, an attribute byte for that field must be supplied in the TIOA by the application program.

  number
  : is an absolute displacement (relative to zero) from the beginning of the map being defined.

  (line,column)
  : are line and column specifications (relative to one) within the map being defined.

ATTRB=
: is applicable only to fields to be displayed on a 3270 and specifies device-dependent characteristics and attributes, such

as the capability of a field to receive data or the intensity to be used when the field is output. If the ATTRB operand is specified within a group of fields, it must be specified in the first field entry. A group of fields appears as one field to the 3270. Therefore, the ATTRB specification refers to all of the fields in a group as one field rather than as individual fields.

ASKIP
: indicates that data cannot be keyed into the field and causes the cursor (current location pointer) to automatically skip over the field.

PROT
: is similar to ASKIP, but no automatic skipping of the field by the cursor occurs.

UNPROT
: indicates that data can be keyed into the field.

NUM
: ensures that the data entry keyboard is set to numeric shift for this field unless the operator presses the alpha shift key, and prevents entry of nonnumeric data if the Keyboard Numeric Lock feature is installed.

BRT
: specifies that a high-intensity display of the field is required.

NORM
: specifies that the field intensity is to be normal.

DRK
: specifies that the field is nonprint/nondisplay.

DET
: specifies that the field is potentially pen-detectable (see programming note 3).

IC
: indicates that the cursor is to be placed in the first position of this field. The IC attribute for the last field for which it is specified in a map is the one that takes effect. If not specified for any fields in a map, the default location is zero. Specifying IC with ASKIP or PROT causes the cursor to be placed in an unkeyable field.

FSET
: specifies that the modified data tag (MDT) for this field should be set when the field is sent to a terminal (see programming note 4).

Either of two sets of defaults may apply when a field to be displayed on a 3270 is being defined but not all parameters are specified. If no ATTRB parameters are specified, ASKIP and NORM are assumed. If any parameter is specified, UNPROT and NORM are assumed for that field unless overridden by a specified parameter.

If the ATTRB operand is specified when fields are combined under a group name, it must be included in the first DFHMDF macro instruction for the group. A group of fields appears as one field to BMS; therefore, the attribute byte refers to all of

the fields in a group as one field rather than to each field individually.

LENGTH=number
indicates the length (from 1 to 256 bytes) of this field. This specified length should be the maximum length required for application-program data to be entered into the field; it should not include the one-byte attribute indicator appended to the field by CICS/VS for use in subsequent processing. LENGTH can be omitted if PICIN or PICOUT is specified but is required otherwise. (See also programming note 5.)

JUSTIFY=
indicates the field justifications for input operations. This operand is ignored for VTAM-supported 3600 terminals, as input mapping is not available.

LEFT
specifies that the data in the input field is left-justified.

RIGHT
specifies that the data in the input field is right-justified.

BLANK
specifies that blanks are to be inserted in any unfilled positions in an input field.

ZERO
specifies that zeros are to be inserted in any unfilled positions in an input field.

LEFT and RIGHT are mutually exclusive, as are BLANK and ZERO. If certain parameters are specified but others are not, assumptions are made as follows:

Specified	Assumed
LEFT	BLANK
RIGHT	ZERO
BLANK	LEFT
ZERO	RIGHT

If the JUSTIFY operand is omitted, but the NUM attribute is specified, RIGHT and ZERO are assumed. If JUSTIFY is omitted, but attributes other than NUM are specified, LEFT and BLANK are assumed.

Note: If a field is initialized by an output map or contains data from any other source, data that is keyed as input may not be justified and the additional data may remain in the field.

INITIAL='any user information'
is used to specify constant or default data for an output field. For fields with the DET attribute, initial data that begins with a blank character, ">", or "?" should be supplied. The number of characters that can be specified in the INITIAL operand is restricted to the continuation limitation of the assembler to be used or to the value specified in the LENGTH operand (whichever is the smaller).

GRPNAME=group name
is the one- to seven-character name used to generate symbolic storage definitions and to combine specific fields under one group name. The fields composing a group must be contiguous, and the same group name must be specified for each field that

is to belong to the group. A field name must be specified for every field that belongs to the group, and the POS operand must also be specified to ensure the fields are contiguous. If this operand is specified, the OCCURS operand (below) cannot be specified. (See programming note 6.)

OCCURS=number
specifies that the indicated number of entries for the field are to be generated in a map and that the map definition is to be generated in such a way that the fields are addressable as entries in a matrix or an array. This permits several data fields to be addressed by the same name (subscripted, of course) without generating a unique name for each field (see programming note 6). OCCURS and GRPNAME are mutually exclusive; that is, OCCURS cannot be used when fields have been defined under a group name. If this operand is omitted, a value of 1 is assumed.

PICIN='value'
specifies a picture to be applied to an input field in an IN or INOUT map; this picture serves as an editing specification which is passed to the application program, thus permitting the user to exploit the editing capabilities of ANS COBOL or PL/I (see programming note 7). BMS checks 'value' to ascertain that the specified characters are valid for input for the language of the map. However, no validity checking of the input data is performed by BMS or the high-level language when the map is used, so any desired checking must be performed by the application program. The length of the data associated with 'value' should be the same as that specified in the LENGTH operand if LENGTH is specified. If both PICIN and PICOUT (see below) are used, an error message is produced if their calculated lengths do not agree; the shorter of the two lengths is used. If PICIN or PICOUT is not coded for the field definition, a character definition of the field is automatically generated regardless of other operands that are coded, such as ATTRB=NUM.

PICOUT='value'
is similar to PICIN, except that a picture to be applied to an output field in the OUT or INOUT map is generated.

Programming Notes:

1. If no name is specified for a field, an application program cannot access the field map to change its attributes or alter its contents. For an output map, omitting the field name may be appropriate when the INITIAL operand is used to specify field contents. If a field name is specified and the map that includes the field is used in a mapping operation, any data supplied by the user overlays data supplied by initialization (unless DATA=NO is specified or assumed by default).

2. The POS operand defines the location of fields in a map. The location of data on the output medium is dependent on DFHMDI macro parameters as well.

   For each field definition (DFHMDF macro instruction), the first position is reserved for an attribute byte. When supplying data for input mapping from non-3270 devices, the actual input data must allow space for this attribute byte. Input data must not start in column 1 but may start in column 2.

3. The first character of a 3270 pen-detectable field must be a "?", ">", or blank. If the first character is a blank, the field is a light pen attention field. (See 3270 Information

Display System Component Description for the functions of the other characters and for cther requirements of pen-detectable fields.) A field for which BRT is specified is potentially pen-detectable to the 3270 but is not recognized as such by BMS unless DET is also specified. DET and DRK are mutually exclusive options. For input map fields, DET and NUM are the only valid

options; all others are ignored. A one-byte reserved area of each input DET field is set to X'00' when the field is unselected, or to X'FF' when the field is selected. No other data is supplied.

4. Specification of FSET causes the 3270 to treat the field as though it has been modified. On a subsequent read from the terminal, this field is read, whether or not it has been modified. The MDT remains set until the field is rewritten without ATTRB=FSET or an output mapping request (for example, DFHMSD CTRL=FRSET or DFHBMS CTRL=FRSET) causes the MDT to be reset.

5. The map dimensions specified in the SIZE operand of the DFHMDI macro instruction defining a map may be smaller than the actual page size or screen size as defined for the terminal. The LENGTH specification in a DFHMDF macro instruction cannot cause the map-defined boundary on the same line to be exceeded. That is, the length declared for a field cannot exceed the number of positions available from the starting position of the field to the final position of the map-defined line. For example, given an 80-position page line, the following map definition and field definition are valid:

```
DFHMDI  SIZE=(2,40),...
DFHMDF  POS=22,LENGTH=17,...
```

But the following definitions are not acceptable:

```
DFHMDI  SIZE=(2,40),...
DFHMDF  POS=22,LENGTH=30,...
```

6. As an example, assume the following map definition (where X in LANG=X is ASM, COBOL, or PLI, and XX in DATA=XX is FIELD or BLOCK, as detailed below):

```
MAPX    DFHMSD  TYPE=DSECT,MODE=INOUT,LANG=X,DATA=XX
MAP     DFHMDI  LINE=1,COLUMN=1,SIZE=(12,80)
F1      DFHMDF  POS=0,LENGTH=80
F2      DFHMDF  POS=100,LENGTH=10,OCCURS=10
F3      DFHMDF  POS=220,LENGTH=80
F4      DFHMDF  POS=650,LENGTH=20,GRPNAME=GRP2
F5      DFHMDF  POS=670,LENGTH=20,GRPNAME=GRP2
        DFHMSD  TYPE=FINAL
```

For Assembler language:

If LANG=ASM and DATA=FIELD, the following DSECT is generated:

```
MAPI    DS      0C .                    INPUT MAP ORIGIN
MAPO    DS      0C .                    OUTPUT MAP ORIGIN

F1L     DS      H  .                  INPUT DATA FIELD LENGTH
F1F     DS      0C .                  DATA FIELD FLAG
F1A     DS      C .                   DATA FIELD ATTRIBUTE
F1I     DS      0CL80 .             INPUT DATA FIELD
F1O     DS      CL80 .              OUTPUT DATA FIELD

F2D     DS      0C .                    FIRST OCCURRING FIELD
F2L     DS      H  .                    INPUT DATA FIELD LENGTH
F2F     DS      0C .                    DATA FIELD FLAG
F2A     DS      C .                     DATA FIELD ATTRIBUTE
F2I     DS      0CL10 .             INPUT DATA FIELD
F2O     DS      CL10 .              OUTPUT DATA FIELD
```

```
F2N      EQU    * .                          NEXT OCCURRING FIELD
           ORG    F2D+10*(3+10)          ALLOCATE OCCURRING FIELD SPACE

F3L      DS     H   .                        INPUT DATA FIELD LENGTH
F3F      DS     0C .                         DATA FIELD FLAG
F3A      DS     C .                          DATA FIELD ATTRIBUTE
F3I      DS     0CL80 .                    INPUT DATA FIELD
F3O      DS     CL80 .                     OUTPUT DATA FIELD

*  START NEW DATA GROUP GRP2
GRP2L    DS     H   .                        INPUT GROUP FIELD LENGTH
GRP2F    DS     0C .                         GROUP FIELD FLAG
GRP2A    DS     CL1 .                        GROUP FIELD ATTRIBUTE
GRP2I    DS     0C .                         INPUT GROUP FIELD ORIGIN
GRP2O    DS     0C .                         OUTPUT GROUP FIELD ORIGIN


F4I      DS     0CL20 .                    INPUT DATA FIELD
F4O      DS     CL20 .                     OUTPUT DATA FIELD

F5I      DS     0CL20 .                    INPUT DATA FIELD
F5O      DS     CL20 .                     OUTPUT DATA FIELD

MAPE     EQU    *       .                    END OF MAP DEFINITION
           ORG
MAPXT    EQU    *       .                    END OF MAP SET DEFINITION
* * *  END OF MAP DEFINITION  * * *
```

If LANG=ASM and DATA=BLOCK, the following DSECT is generated:

```
MAPI     DS     0C .                         INPUT MAP ORIGIN
MAPO     DS     0C .                         OUTPUT MAP ORIGIN

F1F      DS     0C .                        DATA FIELD FLAG
F1A      DS     C .                         DATA FIELD ATTRIBUTE
F1I      DS     0CL80 .                    INPUT DATA FIELD
F1O      DS     CL80 .                     OUTPUT DATA FIELD

           DS     CL19
F2D      DS     0C .                         FIRST OCCURRING FIELD
F2F      DS     0C .                         DATA FIELD FLAG
F2A      DS     C .                          DATA FIELD ATTRIBUTE
F2I      DS     0CL10 .                    INPUT DATA FIELD
F2O      DS     CL10 .                     OUTPUT DATA FIELD
F2N      EQU    * .                          NEXT OCCURRING FIELD
           ORG    F2D+10*(1+10)          ALLOCATE OCCURRING FIELD SPACE

           DS     CL10
F3F      DS     0C .                         DATA FIELD FLAG
F3A      DS     C .                          DATA FIELD ATTRIBUTE
F3I      DS     0CL80 .                    INPUT DATA FIELD
F3O      DS     CL80 .                     OUTPUT DATA FIELD

           DS     CL349
*  START NEW DATA GROUP GRP2
GRP2F    DS     0C .                         GROUP FIELD FLAG
GRP2A    DS     CL1 .                        GROUP FIELD ATTRIBUTE
GRP2I    DS     0C .                         INPUT GROUP FIELD ORIGIN
GRP2O    DS     0C .                         OUTPUT GROUP FIELD ORIGIN

F4I      DS     0CL20 .                    INPUT DATA FIELD
F4O      DS     CL20 .                     OUTPUT DATA FIELD

F5I      DS     0CL20 .                    INPUT DATA FIELD
```

```
F5O      DS      CL20 .                    OUTPUT DATA FIELD

MAPE     EQU  *        .                      END OF MAP DEFINITION
           ORG
MAPXT    EQU   *       .                      END OF MAP SET DEFINITION
* * * END OF MAP DEFINITION * * *
```

For ANS COBOL:

If LANG=COBOL and DATA=FIELD, the following DSECT is generated:

```
01  MAPI.
    02  F1L    COMP    PIC  S9(4).
    02  F1A    PICTURE X.
    02  FILLER REDEFINES F1A.
      03  F1F   PICTURE X.
    02  F1I  PIC X(80).
    02  F2D OCCURS 10 TIMES.
      03  F2L   COMP  PIC  S9(4).
      03  F2A   PICTURE X.
      03  F2I PIC X(10).
      03  FILLER PIC X.
    02  F3L    COMP    PIC  S9(4).
    02  F3A    PICTURE X.
    02  FILLER REDEFINES F3A.
      03  F3F   PICTURE X.
    02  F3I  PIC X(80).
    02  FILLER PIC X.
    02  GRP2L    COMP  PIC  S9(4).
    02  GRP2A    PICTURE X.
    02  FILLER REDEFINES GRP2A.
      03  GRP2F   PICTURE X.
    02  GRP2I.
      03  F5I  PIC X(20).
      03  F4I  PIC X(20).
      03  FILLER PIC X.
01  MAPO REDEFINES MAPI.
    02  FILLER PICTURE X(3).
    02  F1O  PIC X(80).
    02  FILLER PIC X.
    02  DFHMS1 OCCURS 10 TIMES.
      03  FILLER PICTURE X(2).
      03  F2F  PICTURE X.
      03  F2O  PIC X(10).
      03  FILLER PIC X.
    02  FILLER PICTURE X(3).
    02  F3O  PIC X(80).
    02  FILLER PIC X.
    02  FILLER PICTURE X(3).
    02  GRP2O.
      03  F4O  PIC X(20).
      03  F5O  PIC X(20).
```

If LANG=COBOL and DATA=BLOCK, the following DSECT is generated:

```
01  MAPI.
    02  F1A    PICTURE X.
    02  FILLER REDEFINES F1A.
      03  F1F     PICTURE X.
    02  F1I  PIC X(80).
    02  FILLER PICTURE X(19).
    02  F2D OCCURS 10 TIMES.
      03  F2A    PICTURE X.
```

```
      03  F2I  PIC X(10).
    02  FILLER PICTURE X(10).
    02  F3A    PICTURE X.
    02  FILLER REDEFINES F3A.
      03  F3F    PICTURE X.
    02  F3I  PIC X(80).
    02  FILLER PICTURE X(349).
    02  GRP2A     PICTURE X.
    02  FILLER REDEFINES GRP2A.
      03  GRP2F   PICTURE X.
    02  GRP2I.
      03  F4I  PIC X(20).
      03  F5I  PIC X(20).
01  MAPO REDEFINES MAPI.
    02  FILLER PICTURE X(1).
    02  F1O  PIC X(80).
    02  FILLER PICTURE X(19).
    02  DFHMS2 OCCURS 10 TIMES.
      03  F2F  PICTURE X.
      03  F2O  PIC X(10).
    02  FILLER PICTURE X(11).
    02  F3O  PIC X(80).
    02  FILLER PICTURE X(350).
    02  GRP2O.
      03  F4O  PIC X(20).
      03  F5O  PIC X(20).
```

For PLI:

If LANG=PLI and DATA=FIELD, the following DSECT is generated:

```
DECLARE 1 MAPI BASED (BMSMAPER),
     2  F1L    FIXED BINARY (15,0),
     2  F1A    CHARACTER (1),
     2  F1I  CHARACTER (80),
     2  F2D(10),
       3  F2L   FIXED BINARY (15,0),
       3  F2A   CHARACTER (1),
       3  F2I  CHARACTER (10),
     2  F3L    FIXED BINARY (15,0),
     2  F3A    CHARACTER (1),
     2  F3I  CHARACTER (80),
     2  GRP2L   FIXED BINARY (15,0),
     2  GRP2A   CHARACTER (1),
     2  GRP2I,
       3  F4I  CHARACTER (20),
       3  F5I  CHARACTER (20),
     2  FILL0230   CHARACTER (1);
DECLARE 1 MAPO BASED (BMSMAPBR),
     2  DFHMS3 FIXED BINARY (15,0),
     2  F1F CHARACTER (1),
     2  F1O  CHARACTER (80),
     2  DFHMS4(10),
       3  DFHMS5 FIXED BINARY (15,0),
       3  F2F CHARACTER (1),
       3  F2O  CHARACTER (10),
     2  DFHMS6 CHARACTER (2),
     2  F3F CHARACTER (1),
     2  F3O  CHARACTER (80),
     2  DFHMS7 FIXED BINARY (15,0),
     2  GRP2F CHARACTER (1),
     2  GRP2O,
       3  F4O  CHARACTER (20),
       3  F5O  CHARACTER (20),
```

```
     2  FILL0230  CHARACTER (1);
/* END OF MAP DEFINITION */
```

If LANG=PLI and DATA=BLOCK, the following DSECT is generated:

```
DECLARE 1 MAPI BASED (BMSMAPBR),
     2  F1A    CHARACTER (1),
     2  F1I  CHARACTER (80),
     2  DFHMS9 CHARACTER (19),
     2  F2D(10),
       3   F2A   CHARACTER (1),
       3   F2I  CHARACTER (10),
     2  DFHMS12 CHARACTER (10),
     2  F3A    CHARACTER (1),
     2  F3I  CHARACTER (80),
     2  DFHMS14 CHARACTER (349),
     2  GRP2A    CHARACTER (1),
     2  GRP2I,
       3  F4I  CHARACTER (20),
       3  F5I  CHARACTER (20),
     2  FILL0243  CHARACTER (1);
DECLARE 1 MAPO BASED (BMSMAPER),
     2  DFHMS8 CHARACTER (0),
     2  F1F CHARACTER (1),
     2  F1O  CHARACTER (80),
     2  DFHMS10 CHARACTER (19),
     2  DFHMS11(10),
       3  F2F CHARACTER (1),
       3  F2O  CHARACTER (10),
     2  DFHMS13 CHARACTER (10),
     2  F3F CHARACTER (1),
     2  F3O  CHARACTER (80),
     2  DFHMS15 CHARACTER (349),
     2  GRP2F CHARACTER (1),
     2  GRP2O,
       3  F4O  CHARACTER (20),
       3  F5O  CHARACTER (20),
     2  FILL0243  CHARACTER (1);
/* END OF MAP DEFINITION */
```

7. As an example, assume the following map definition is created for reference by an ANS COBOL application program:

```
MAPX    DFHMSD  TYPE=DSECT,LANG=COBOL,MODE=INOUT
MAP     DFHMDI  LINE=1,COLUMN=1,SIZE=(1,80)
F1      DFHMDF  POS=0,LENGTH=30
F2      DFHMDF  POS=40,LENGTH=10,PICOUT='$$$,$$0.00'
F3      DFHMDF  POS=60,LENGTH=6,PICIN='9999V99',PICOUT='ZZ9.99'
        DFHMSD  TYPE=FINAL
```

The following DSECT is generated:

```
    01  MAPI.
        02  F1L     COMP PIC S9(4).
        02  F1A     PICTURE X.
        02  FILLER REDEFINES F1A.
          03  F1F    PICTURE X.
        02  F1I     PIC X(30).
        02  FILLER  PIC X.
        02  F2L     COMP PIC S9(4).
        02  F2A     PICTURE X.
        02  FILLER REDEFINES F2A.
          03  F2F    PICTURE X.
```

```
    02  F2I      PIC X(10).
    02  FILLER   PIC X.
    02  F3L      COMP PIC S9(4).
    02  F3A      PICTURE X.
    02  FILLER REDEFINES F3A.
     03  F3F     PICTURE X.
    02  F3I      PIC 9999V99.
    02  FILLER   PIC X.
01  MAPO REDEFINES MAPI.
    02  FILLER PICTURE X(3).
    02  F1O      PIC X(30).
    02  FILLER   PIC X.
    02  FILLER PICTURE X(3).
    02  F2O      PIC $$$,$$0.00.
    02  FILLER   PIC X.
    02  FILLER PICTURE X(3).
    02  F3O      PIC ZZ9.99.
    02  FILLER   PIC X.
```

ONLINE MAP USE

Online mapping operations are requested by issuing the DFHBMS macro instruction. Parameters provided by the application program indicate whether an input or an output operation is needed, the name of the map to be used by BMS, and other information to control the mapping function. Control is passed to BMS, which performs any required input/output operations through terminal control.

Terminal input, which causes a task to be initiated, is stored in the initial TIOA of the task as a native-mode data stream. By requesting a MAP operation through DFHBMS, the application program is given the capability to map this TIOA into a particular input format. The format of this initial input data must correspond to that of the requested map. Input data to be mapped from a 3270 must contain 3270 device-dependent code (in particular, the data stream must contain an SBA). The data returned from an input mapping operation is in TIOA format; the address of this TIOA is in TCTTEDA.

For an output mapping operation, if data is to be passed from the TIOA of an application program, the application program must have obtained, through storage control, a TIOA large enough to contain the symbolic storage definition of the map being used. Any fields for which data is not to be passed to the mapping operation must be set to nulls (X'00'); this is best achieved through use of the INITIMG=00 operand of the DFHSC TYPE=GETMAIN macro instruction. The first position of a field to be sent must not contain a null; however, if it does, the field will be ignored.

Establishing Addressability to User-Supplied Data

Before issuing the DFHBMS macro instruction, addressability must be established for the data to be passed. If the data is being passed in a TIOA by a terminal-oriented task, the address of this TIOA may be placed at TCTTEDA, or placed in TCAMSIOA and binary zeros placed in TCTTEDA. If the data is being passed by a terminal-oriented task but not in a TIOA, the address of the TIOA-like area of this data must be placed in TCAMSIOA and binary zeros placed in TCTTEDA. If the data is being passed by a nonterminal-oriented task, the address of the TIOA-like area of this data must be placed in TCAMSIOA. TCTTEDA cannot be referred to, because there is no TCTTE associated with this task. If a task attempts to pass addresses in both TCTTEDA and TCAMSIOA, the address in TCTTEDA is used. Since TCTTEDA is altered by BMS, the user should not assume that its contents are unchanged.

Terminal-oriented tasks need not use actual TIOAs. Any task may pass data to BMS in any portion of dynamically acquired storage which looks like a TIOA in all respects, with two possible exceptions:

- The storage class need not be terminal.
- The storage chain address need not refer to a TCTTE or other terminal storage

DFHBMS Macro Instruction

Regardless of the programming language used (Assembler language, American National Standard (ANS) COBOL, or PL/I), the same form of the DFHBMS macro instruction is used to request a mapping operation. In the case of ANS COBOL and PL/I, the CICS/VS preprocessor resolves the macro instruction and expands it into the statements required to invoke the mapping function. A wide variety of options is available with BMS, and a corresponding variety of parameters may be specified in a DFHBMS macro instruction. General usages of the DFHBMS macro instruction are summarized below.

INPUT OPERATIONS

To request BMS services for input operations, a DFHBMS macro instruction of the following format is used:

```
DFHBMS  TYPE={IN|MAP}[,SAVE][,TEXT])
        [,MAP={map name|YES}] | [,MAPADR={symbolic address|YES}]
        [,MAPSET={map set name|YES}] [,MSETADR={symbolic address|YES}]
        [,RDATT=symbolic address]
        [,NORESP=symbolic address]
        [,MAPFAIL=symbolic address]
        [,INVMPSZ=symbolic address]
        [,ERROR=symbolic address]
```

where:

TYPE=
: indicates the BMS service to be performed.

  IN
  : specifies an input mapping operation. Input is accepted from the terminal through a terminal control READ/WAIT request. The input data is then mapped into the TIOA and made available to the application program by placing the TIOA address at TCTTEDA. The fields entered as part of the input data stream are available to the application program under the field names specified in the DFHMDF macro instructions by which they are defined, suffixed with the letter I to correspond to the name generated by CICS/VS in the definition of the area. Data from a VTAM-supported terminal is not mapped, but is left in the TIOA unaltered.

  MAP
  : specifies an input mapping operation similar to IN, except that a terminal control READ/WAIT is not performed. The

application program must have placed the address of an input TIOA containing data to be mapped into TCTTEDA or TCAMSIOA. The data in the TIOA is positioned into a new TIOA using the map specified in the MAP operand of this DFHBMS macro instruction, but no terminal I/O operation occurs. An example of such a TIOA is the initial TIOA given to a transaction upon entering a transaction code. If data is included with the transaction code, the screen must have been formatted previously by another transaction, or the data is not mapped. For a VTAM-supported terminal, mapping is not performed, and the input TIOA is returned unaltered.

SAVE
specifies that the user-supplied data area (address at TCTTEDA or TCAMSIOA) is not to be altered, and a new TIOA is to be acquired for the operation. The address of the new TIOA is returned to the application program in the location in which the original data area was specified (TCTTEDA or TCAMSIOA).

TEXT
can be specified with IN to indicate that uppercase and lowercase characters are contained in the input data stream. This parameter is not valid with MAP, because the input data has already been read into a TIOA.

Note: This parameter is used to override a FEATURE=UCTRAN specification in the DFHTCT macro instruction set up by the system programmer for the input terminal. (See the CICS/VS System Programmer's Reference Manual.)

MAP=

specifies the name of the map to be used when mapping formatted pages.

map name
is the one- to seven-character name of the map within a map set.

YES
indicates that the application programmer has placed the name of the map in TCABMSMN prior to issuing this DFHBMS macro instruction. The name must be left-justified and padded with trailing blanks to eight characters.

If no map set (MAPSET or MSETADR operand) is specified in this DFHBMS macro instruction, the specified map name is taken as the name of the map set.

MAPADR=

specifies the address of the map to be used when mapping formatted pages.

symbolic address
is the one- to seven-character symbolic label that has been assigned to a map coded within an Assembler-language application program.

YES
indicates that the application programmer has placed the address of the map in TCABMSMA prior to issuing this DFHBMS macro instruction.

If MAPADR is specified, neither MAP, MAPSET, nor MSETADR should be used.

MAPSET=
specifies the name of the map set to be used in the mapping operation.

map set name
is the one- to seven-character name of the map set.

YES
indicates that the application programmer has placed the name of the map set in TCAMSMSN prior to issuing this DFHBMS macro instruction. The name must be left-justified and padded with trailing blanks to eight characters.

The map set established by this operand must reside in the CICS/VS program library, and a corresponding entry for the map set must exist in the processing program table (PPT).

If MAPSET is specified, MAP is required.

MSETADR=
specifies the address of the map set to be used in the mapping operation.

symbolic address
is the one- to eight-character symbolic label that has been assigned to the map set coded within the Assembler-language program.

YES
indicates that the application programmer has placed the address of the map set in TCAMSMSA prior to issuing this DFHBMS macro instruction.

MAPSET and MSETADR are mutually exclusive operands. If MSETADR is specified, MAP is required.

RDATT=
specifies the address of a routine to receive control if the operator presses the ATTN key on a 2741 when input is being entered from the terminal in response to a DFHBMS TYPE=IN request. This operand can be specified only if 2741 Read Attention support, an option available under either CICS/DOS/VS or CICS/OS/VS, has been generated into the system (see "2741 Read Attention and Write Break Support" in Chapter 11).

NORESP, MAPFAIL, INVMPSZ, and ERROR
are used to test the response of BMS to this request for BMS services. These operands can be specified in this macro instruction or in a DFHBMS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for BMS Services."

OUTPUT OPERATIONS

The result of one or more requests for BMS output services is a logical message. By definition, a logical message is the output produced from a sequence of BMS output requests, all of which have the same disposition (OUT, STORE, or RETURN), either implied or specified. A logical message is terminated by the occurrence of any one of the following:

- A change in disposition
- A DFHBMS TYPE=PAGEOUT request
- Termination of the applicatcn program

Cumulative Page Building with Mapping

To request BMS services for output mapping operations, the application program issues a DFHBMS TYPE=PAGEBLD macro instruction. This causes the data in the area defined by a specified symbolic description map to be mapped according to the physical map. The mapped data is positioned within an area large enough to contain one page of output. The application programmer issues another DFHBMS TYPE=PAGEBLD macro instruction to map and position the next portion of the output page. This mapping operation continues until the application program has completed the message to be sent to the terminal.

Because of CICS/VS terminal paging facilities, the application programmer need not keep track of when a page is full. It is only necessary to include the OFLCW operand in the DFHBMS TYPE=PAGEBLD macro instructions to cause BMS to transfer control to an overflow routine (which the programmer must provide) when a page of data cannot contain the data to be mapped.

The format of the DFHBMS TYPE=PAGEBLD macro instruction is as follows:

```
DFHBMS  TYPE=(PAGEBLD[,{OUT[,WAIT]}]  [,SAVE][,ERASE][,LAST])
                       {STORE      }
                       {RETURN     }
        [,IOTYPE={IMMED   }]
                 {DELAY   }
        [,LDC={mnemonic}]
              {YES     }
        [,DATA={NO  }]
               {YES }
               {ONLY}
        [,MAP={map name}]
              {YES     }
        [,MAPSET={mapset name}][,MSETADR={symbolic address}]
                 {YES        }           {YES             }
        [,CTRL=([PRINT][,{L40 }][,FREEKB][,ALARM][,FRSET])]
                         {L64 }
                         {L80 }
                         {HONEOM}
         [,OFLOW=symbolic address]
         [,PROPT=NLEOM]
         [,CURSOR={number}]
                  {YES   }
         [,REQID={'prefix'}]
                 {YES     }
         [,FMHPARM={parameter}]
                   {YES      }
         [,WRBRK=symbolic address]  CICS/OS/VS only
         [,NORESP=symbolic address]
         [,TSIOERR=symbolic address]
         [,INVREQ=symbolic address]
         [,INVLDC=symbolic address]
         [,RETPAGE=symbolic address]
         [,INVMPSZ=symbolic address]
         [,IGREQID=symbolic address]
         [,ERROR=symbolic address]
```

where:

TYPE=
indicates the general output functions required.

Combining of Output

PAGEBLD
indicates that one page of data is to be accumulated and formatted from data submitted through multiple PAGEBLD requests. In each PAGEBLD request, a map defines the number of lines and columns that the data is to occupy on the page. When a page is complete, as defined by one or more maps, it is written according to an OUT, STORE, or RETURN disposition (see below).

If neither PAGEBLD nor TEXTBLD (explained below) is specified in an output request, data is processed and written as output in a single operation; no combining of data is performed.

Disposition

OUT
indicates that the output is to be written to the originating terminal when the page is complete, if the originating terminal is to receive the output.

The application program may issue DFHSC TYPE=FREEMAIN, RELEASE=ALL requests when using DFHBMS TYPE=OUT only under the following conditions:

- Prior to any DFHBMS requests
- Between a DFHBMS TYPE=PAGEOUT and a subsequent BMS request
- After a DFHBMS TYPE=PURGE request

WAIT
indicates that BMS is to wait until all output operations are complete before returning control to the application program. WAIT must be specified with every output request except the following:

- The last output request prior to task termination
- The last output request prior to an input operation
- The last output request prior to issuing a DFHBMS TYPE=PAGEOUT macro instruction that precedes task termination or an input operation

STORE
indicates that the output is to be placed in temporary storage to be displayed in response to paging commands entered by the terminal operator (for more information about these commands, see the CICS/VS Terminal Operator's Guide). If STORE is specified, with a REQID that is defined in the Temporary Storage Table (TST), CICS/VS provides message recovery for logical messages if the task has reached logical end.

RETURN
indicates that the complete page(s) is to be returned to the application programmer (see programming note 1). The

application program regains control (1) immediately following the BMS instruction if the current page is not yet completed, or (2) at an alternative entry point specified through the RETPAGE operand of this macro instruction if one or more pages have been ccmpleted.

If no disposition is specified, the output is sent to the originating terminal. Once the disposition has been established for a logical message, it is not necessary to repeat the disposition for that logical message. Any change of disposition specified while in the process of building a logical message forces that logical message to completion with its original disposition. Then a new logical message is started with the new disposition. The disposition parameter is handled differently under DFHBMS TYPE=ROUTE (see "Disposition and Message Routing").

SAVE
specifies that the user-supplied data area (address at TCTTEDA or TCAMSIOA) is to be saved. The contents of these areas are not restored and must be reset for subsequent BMS requests.

ERASE
specifies that a 3270 buffer or 3604 screen is to be erased before this page of output is displayed. A 3284/3286 buffer would contain meaningless data from prior messages if all positions are not filled with current data.

Note: If CTRL= is not specified in a request for a 3270, an appropriate WCC for the 3270 must be placed in TIOACLCR before the request is issued.

LAST
signals CICS/VS that this is the last output for a transaction and, therefore, the end of a bracket operation. This operand is meaningful only for VTAM-supported terminals and is applicable only when OUT is the specified disposition.

IOTYPE=
specifies when the output operation is to be started. This operand is meaningful only for VTAM-supported terminals and is applicable only when OUT is the specified disposition.

IMMED
causes the output operaticn to be started immediately. In cases where WAIT may be coded separately from DFHBMS TYPE=PAGEBLD, overlap of terminal I/O operations is possible by using IMMED and coding the WAIT separately.

DELAY
can cause the output operation to be started only when TCP is next dispatched.

If this operand is omitted, IOTYPE defaults to the transaction option specified by TIOTYPE in the DFHPCT TYPE=ENTRY macro instruction. A task using deferred write or message protection will override this parameter and not be done until sync point (DFHSP) or task terminated.

LDC=
specifies the mnemonic to be used by CICS/VS to determine the logical device code that is to be used for the PAGEBLD operation

and transmitted in the function management header (FMH) to the logical unit. This operand is meaningful only for VTAM-supported terminals with LDC support.

mnemonic
    is the two-character mnemonic used to determine the

appropriate LDC numeric value. The mnemonic represents a LDC entry in the DFHTCT TYPE=LDC macro instruction.

YES
indicates that the application program has placed the LDC mnemonic in TCAMSLDM.

If this operand is omitted for a VTAM-supported terminal, a default LDC is chosen. This default LDC is the first in the local LDC table for that logical unit. The numeric value for this LDC is inserted into the FMH. The system table is scanned to find the page size, page status, and device type associated with the default LDC; BMS then uses these values to format the message. If no local table exists, a null value (X'00') is inserted into the FMH; the page size used by BMS is the value specified in the PGESIZE operand of the DFHTCT TYPE=TERMINAL macro instruction, and a default device type of 3604 is assumed. The page size specified in this macro instruction applies as a default to all terminals belonging to the logical unit. If the PGESIZE operand is omitted, a default of (1,40) is assumed.

DATA=
indicates one of the following three output mapping data selection modes.

NO
specifies that only default data is to be written from the selected map.

YES
indicates that default data from the selected map is to be merged with data placed in the TIOA by the application program.

ONLY
specifies that only application-program-supplied data is to be written.

This operand is valid only when mapping is used. If it is omitted, DATA=NO is assumed.

MAP=
specifies the name of the map to be used when mapping formatted pages.

map name
is the one- to seven-character name of the map within a map set.

YES
indicates that the application programmer has placed the name of the map in TCABMSMN prior to issuing this DFHBMS macro instruction. The name must be left-justified and padded with trailing blanks to eight characters.

MAPSET=
specifies the name of the map set to be used in the mapping operation.

map set name
is the one- to seven-character name of the map set.

YES
indicates that the application programmer has placed the name of the map set in TCAMSMSN prior to issuing the DFHBMS

macro instruction. The name must be left-justified and padded with trailing blanks to eight characters.

The map set established by this operand must reside in the CICS/VS program library, and a corresponding entry for the map set must exist in the processing program table (PPT).

MSETADR=
specifies the address of the map set to be used in the mapping operation.

symbolic address
is the one- to eight-character symbolic label that has been assigned to the map set coded within the Assembler-language application program.

YES
indicates that the application programmer has placed the address of the map set in TCAMSMSA prior to issuing this DFHBMS macro instruction.

MAPSET and MSETADR are mutually exclusive operands.

CTRL=
is used to specify device characteristics related to terminals of the 3270 Information Display System. CTRL=ALARM is valid also for VTAM terminals; all other parameters for CTRL= are ignored.

PRINT
must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed. This operand is ignored for 3275s without the Printer Adapter feature and for 3277s.

L40,L64,L80,HONEOM
are mutually exclusive options that control the line length on the printer. L40, L64, and L80 force a carrier return/line feed after 40, 64, or 80 characters, respectively. HONEOM causes the printer to honor all new-line (NL) characters and the first end-of-message (EM) character that appear in displayable fields of the data stream. If the latter option is specified, the application program must insert the NL and EM characters into the data stream. If the NL character is omitted, a carrier return/line feed occurs at the physical end of the carriage. If the EM character is omitted, printing stops at the end of the 3270 buffer.

FREEKB
specifies that the keyboard should be unlocked after this map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM
activates the 3270 audible alarm feature. For VTAM terminals supporting functional map headers (FMH), ALARM signals BMS to set the alarm flag in the FMH.

FRSET
indicates that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before any map data is written to the buffer. This allows the DFHMDF

ATTRB specification for the requested map to control the final status of any fields written or rewritten in response to a DFHBMS macro instruction.

OFLOW=symbolic address
specifies the symbolic address of a routine to which control is to be transferred if the mapped data does not fit on the current page (see "PAGEBLD Overflow Processing").

PROPT=NLEOM
is applicable to output to 3284, 3286, and 3275 with the printer adapter feature. This parameter is restricted to assembler language programs. If it is to be used, it must be coded with the first DFHBMS request for a logical message.

When this parameter is used, data for these terminals is built with new line characters, as with any other hard copy device. An end-of-message character (EOM) is placed at the end of the data. As the data is printed a new line character in the data terminates printing of a line, and the EOM character terminates printing of the data.

If this parameter is not used and one of the CTRL parameters L40, L64, L80, or HONEOM is not used, the data is printed without regard to the presence of new line or EOM characters in the data; the entire buffer is printed in lines the width of the printer.

The following restrictions apply when using this parameter: Buffer updating and attribute modification of fields previously written into the buffer are not allowed. BMS issues an ERASE with every write to the terminal.

The new line character occupies a buffer position. A number of buffer positions, equivalent to the value of the PGESIZE parameter of the DFHTCT macro for that terminal, is unavailable for data. This may cause data to wrap around in the buffer; if this occurs, it is necessary to reduce the page size specifications in the TCT for the terminal.

CURSOR=
is used to position the cursor upon completion of a write operation to a 3270 device.

number
is an integer indicating a particular position relative to zero on the screen; the range of values that may be specified depends upon the screen size of the 3270 being used.

YES
indicates that a value indicating the desired cursor position has been placed in TCABMSCP.

An alternate method may be used to dynamically position the cursor on the output screen. This method is called symbolic cursor positioning (SCP). SCP allows a field in the TIOA to be marked, symbolically, such that the cursor is placed under the first data byte of the field on the output screen.

Requirements for SCP use are as follows:

- MODE=INOUT must be specified on the DFHMSD macro for maps and DSECTs which will be used with SCP.

- CURSOR=YES must be specified on the DFHBMS macro.
- Field TCABMSCP must be initialized with hexadecimal Fs; for example, MVC TCABMSCP,=X'FFFF'.
- The length field, suffix "L", associated with the field under which the cursor is to be placed must be initialized with hexadecimal Fs. For example, MVC FIELD3L,=X'FFFF'.

The remainder of the TICA may be built as desired by the user. SCP is operable only for devices which allow cursor placement to be performed independent of data placement; for example, 3604 and 3270. SCP specification is ignored for other devices.

REQID=
specifies the prefix to be used with the Temporary Storage identification. The identification (including the prefix) is used by CICS/VS when attempting message recovery.

BMS message recovery is provided for a logical message only if the STORE operand is specified in the BMS output request and if the logical end of task has been reached.

Only one prefix can be specified for each logical message. If the REQID operand is not specified, CICS/VS assigns the prefix ** (two asterisks).

'prefix'
indicates the alphanumeric prefix to be used as the first two characters of a Temporary Storage identification.

YES
indicates that the prefix has been stored at TCAMSRID.

FMHPARM=
specifies information to be included in a function management header (FMH) being transmitted to a 3650 logical unit. Refer to the CICS/VS Advanced Communication Guide for details on the FMH and on 3650 logical units.

This operand applies only to 3650 logical units with outboard formatting. It specifies the name of the map to be used with this BMS request.

parameter
specifies the eight-character name of the map.

YES
indicates that the map name has been stored in the eight-character TCAMSFMP field.

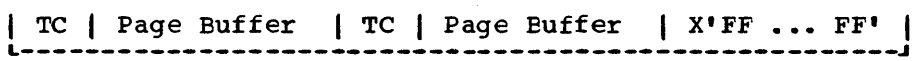
WRBRK=symbolic address
specifies the symbolic address of the routine to receive control when the ATTN key on a 2741 is pressed during the actual write to the terminal. This operand is operative when 2741 Write Break support has been generated into CICS/VS (available only under OS/VS) and when the task would have normally regained control. It is not operative when TYPE=STORE or TYPE=RETURN is specified.

NORESP, TSIOERR, INVREQ, INVIDC, RETPAGE, INVMPSZ, IGREQID, and ERROR
are used to test the BMS response to this request for BMS services. These operands can be specified in this macro instruction or in a DFHBMS TYPE=CHECK macro instruction. The

meaning of each operand is discussed in detail under "Test Response to a Request for BMS Services."

Programming Note:

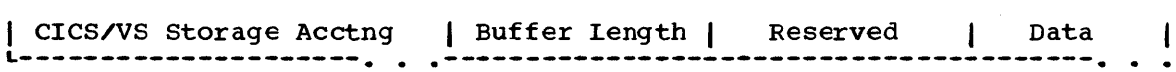
1. Whenever one or more pages have been completed and the programmer has specified TYPE=RETURN, TCAMSRLA contains the address of a list of completed pages. Since more than one page of output may result from a single BMS output request, there may be more than one entry in the list for a given terminal type. The multiple entries for a terminal type immediately follow one another in the list. The list is laid out as follows:



4 bytes 4 bytes 4 bytes

TC = Terminal code (see "Terminal Code (TC) Table")

The page buffer pointer points to an area of USER-type storage which has a 12-byte prefix similar to that of a terminal input/output area (TIOA):



8 bytes 2 bytes 2 bytes x bytes

At this point, page buffers are on the user's storage chain and are disassociated from BMS control blocks; it is therefore the user's responsibility to release page buffers when they are no longer needed. The storage containing the list of buffers should not be freed by the programmer; it is the intention of BMS to reduce processing time by reusing the list. This list will be altered by the next BMS request. Therefore, the programmer must save the contents before issuing the next BMS request.

When terminals of the 3270 Information Display System are used, the write control character (WCC) containing the CTRL specification can be found at TIOACLCR in the page buffer after addressability to the area has been established. (TIOACLCR is a defined field in DFHTIOA and is addressable if TIOABAR is loaded into the buffer address.)

PAGEBLD Overflow Processing

Overflow occurs when the number of lines in the requested map plus the number of lines in the largest trailer map in the map set (if there are any trailer maps) is greater than the number of lines remaining in the page being built for the terminal involved in an output operation. For VTAM terminals having LDC support, pages are accumulated individually by LDC mnemonic. Therefore, overflow may occur at end of page for each different LDC mnemonic used in different BMS requests. The LDC mnemonic is passed to the user's overflow routine in TCAMSLDM, and the LDC numeric value is passed in TCAMSLDC. PAGEBLD overflow can occur on a logical message being built for a ROUTE environment. If the ROUTE environment was created with a route list containing more than one LDC mnemonic, then the returned LDC mnemonic and numeric value is the first LDC mnemonic resolved in the route list.

The routine to which control is transferred must be in the application program, but no special considerations apply. The data

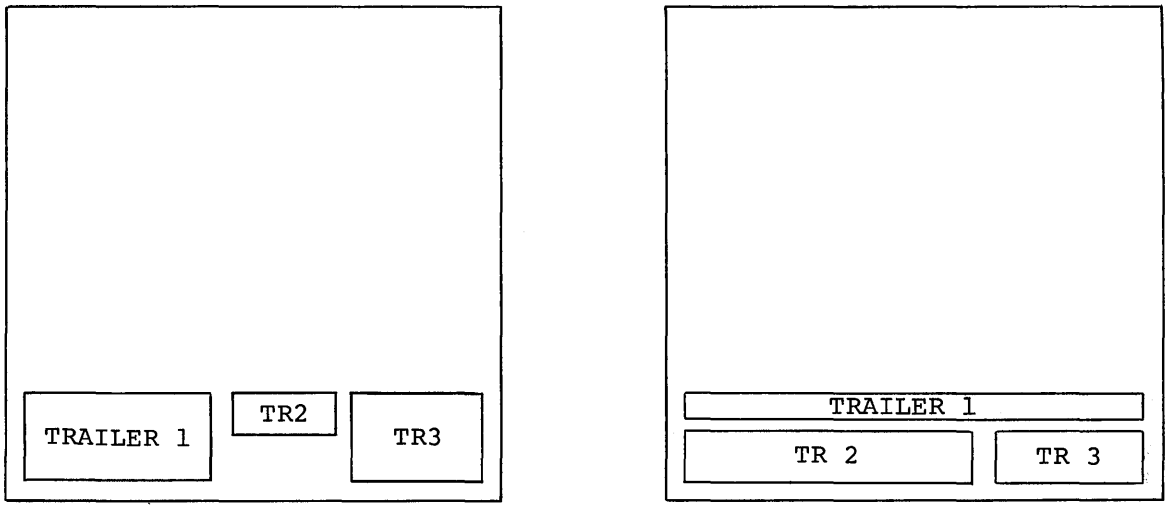
which was to have been mapped, but which caused the overflow, is not mapped by BMS and remains unaltered in the TIOA.

If a DFHBMS TYPE=ROUTE macro instruction has not been previously issued, there is only one destination. If a DFHBMS TYPE=ROUTE macro instruction has been issued, the logical message is probably being built for a multiple-destination environment. Since the application programmer has the capability of concurrently building pages for terminals that have different-sized output, overflow may occur at different times for different terminal groups. The overflow routine gets control every time any one of the destinations or groups of destinations encounters an overflow condition. The application program overflow routine must determine which destination or group of destinations has encountered the overflow.

Upon return to the application program from a DFHBMS TYPE=ROUTE macro instruction, a count (relative to one) of the number of destinations or groups of destinations is available in TCAMSOCN. This overflow control count tells the application programmer how many overflow control areas (for example, accumulators) he may want to keep. Whenever the overflow routine gets control, TCAMSOCN indicates the relative overflow control number of the destination that has encountered the overflow. This number indicates which control area should be output, perhaps through one or more trailer maps. In addition to the relative control count, BMS returns the current page number for the destination that has encountered the overflow. This page number is located at TCAMSPGN.

To place trailer data on a page, the programmer codes DFHBMS TYPE=PAGEBLD request(s) to process the trailer data. The map(s) used

to format the data must contain TRAILER=YES so that the amount of space on the page to reserve for overflow can be calculated. More than one trailer map may be placed on a page. There should be a dummy trailer map (not otherwise used) in the map set specifying the number of lines to be reserved for trailer data if no single trailer map extends over the total number of lines required for trailer data (see Figure 10-1). Maps used to map trailer data may contain JUSTIFY=LAST to force their placement at the bottom of the page. If the programmer tries to place more lines of trailer data on the page than are available, that trailer data is placed on a separate page by itself. Still another page is built to continue mapping with or without a header map.



No dummy trailer required. Dummy trailer is required.

Figure 10-1. Use of Trailer Maps in PAGEBLD Mapping Operations

To place header data on a page, the programmer codes DFHBMS TYPE=PAGEBLD request(s) to process the header data. The map(s) used to map header data must specify JUSTIFY=FIRST to complete processing of the previous page if that has not been done, and to begin a new page. (JUSTIFY=FIRST is ignored if BMS is positioned at the top of a new page.) If the programmer tries to place more header data on the page than the page can contain, multiple pages are created.

When all trailer and/or header data has been processed, the programmer must reissue the DFHBMS request that caused the overflow, since this data has not yet been mapped for all destinations.

It is important to recognize that BMS maintains the overflow environment for as long as the application program issues BMS requests using maps defined as headers or trailers. The first use of a map that is not defined as a header or trailer terminates overflow processing. Presumably, this coincides with reissuing the request that caused the overflow.

If the user does not specify an overflow routine while issuing PAGEBLD requests, no overflow occurs and new pages will be forced automatically. If a header is to be placed on the first page and a trailer on the last, the OFLOW parameter would not be used.

A general overview of overflow processing is given in the flowchart in Figure 10-2.

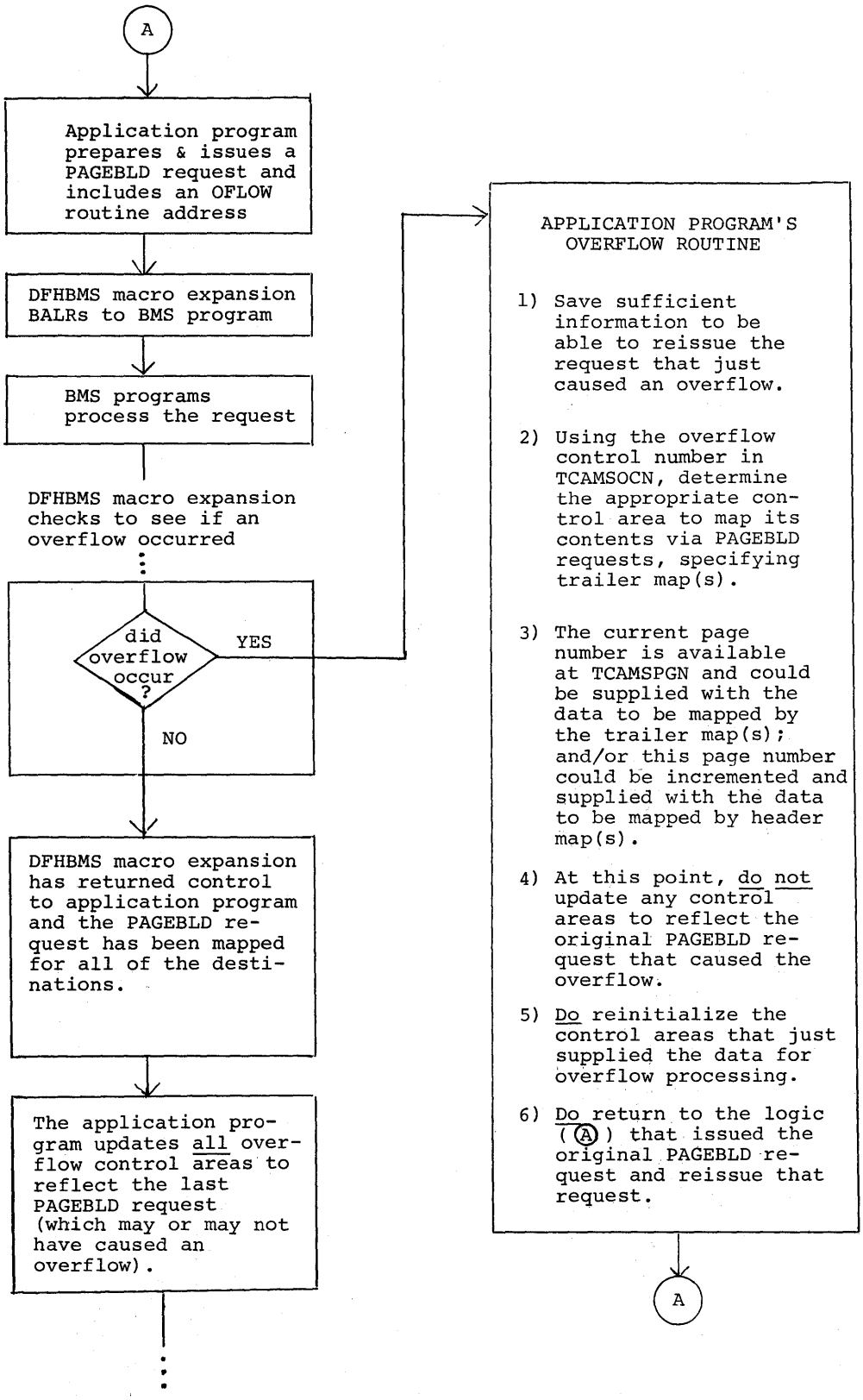


Figure 10-2. Overflow Processing by Application Programs under BMS

Cumulative Page Building without Mapping

To request the building of pages of data without the use of maps, the application program issues DFHBMS TYPE=TEXTBLD macro instructions. These macro instructions cause BMS terminal paging to create pages containing application-program-supplied text data. The length of the data each macro instruction is to process must be supplied in TIOATDL, prior to issuing the macro instruction. Completion of a logical message is signaled by a DFHBMS TYPE=PAGEOUT macro instruction. The beginning and ending of pages are handled by BMS and need be of no concern to the application program.

The format of the DFHBMS TYPE=TEXTBLD macro instruction is as follows:

```
DFHBMS  TYPE=(TEXTBLD[,{OUT[,WAIT]}]  [,SAVE][,ERASE]
                       {STORE       }
                       {RETURN      }
              [,LAST])
        [,IOTYPE={IMMED   }]
                 {DELAY   }
        [,LDC={mnemonic}]
              {YES     }
        [,HEADER={symbolic address}]
                 {YES             }
        [,TRAILER={symbolic address}]
                  {YES             }
        [,JUSTIFY={FIRST}]
                  {LAST }
                  {nnn  }
                  {YES  }
        [,CTRL=([PRINT][,{L40     }][,FREEKB][,ALARM])]
                         {L64     }
                         {L80     }
                         {HONEOM  }
        [,PROPT=NLEOM]
        [,CURSOR={number}]
                 {YES   }
        [,REQID={'prefix'}]
                {YES      }
        [,FMHPARM={parameter}]
                  {YES      }
        [,WRBRK=symbolic address]  CICS/OS/VS only
        [,NCRESP=symbolic address]
        [,TSIOERR=symbolic address]
        [,INVREQ=symbolic address]
        [,INVLDC=symbolic address]
        [,RETPAGE=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbolic address]
```

where:

TYPE=
: indicates the general output functions required.

Combining of Output

TEXTBLD
indicates that (1) one page of output is to be formed from data submitted through multiple TEXTBLD requests, or (2) multiple pages of output are to be formed from one TEXTBLD request. When TEXTBLD is specified, no map is used. When no more data can fit on a page, the page is written according to the OUT, STORE, or RETURN disposition (see below), and another page is started if necessary.

If neither TEXTBLD nor PAGEBLD (explained earlier) is specified in an output request, data is processed and written as output in a single operation; no combining of data is performed.

Disposition

OUT
indicates that the output is to be written to the originating terminal when the page is complete if the originating terminal is to receive the output.

The application program may issue DFHSC TYPE=FREEMAIN, RELEASE=ALL requests when using DFHBMS TYPE=OUT only under the following conditions:

- Prior to any DFHBMS requests
- Between a DFHBMS TYPE=PAGEOUT and a subsequent BMS request
- After a DFHBMS TYPE=PURGE request

WAIT
indicates that BMS is to wait until all output operations are complete before returning control to the application program. WAIT must be specified with every output request except the following:

- The last output request prior to task termination
- The last output request prior to an input operation
- The last output request prior to issuing a DFHBMS TYPE=PAGEOUT macro instruction that precedes task termination or an input operation

STORE
indicates that the output is to be placed in temporary storage to be displayed in response to paging commands entered by the terminal operator (for more information about these commands, see the CICS/VS Terminal Operator's Guide). If STORE is specified with a REQID that is defined in the Temporary Storage Table (TST), CICS/VS provides message recovery for logical messages if the task has reached logical end.

RETURN
indicates that the complete page(s) is to be returned to the application programmer (see programming note 1 under "Cumulative Page Building with Mapping"). The application program regains control (1) immediately following the BMS

instruction, or (2) at an alternative entry point specified through the RETPAGE operand of this macro instruction.

If no disposition is specified, the output is sent to the originating terminal. Once the disposition has been established for a logical message, it is not necessary to repeat the disposition for that logical message. Any change of disposition specified while in the process of building a logical message forces that logical message to completion with its original disposition. Then a new logical message is started with the new disposition. The disposition parameter is handled differently under DFHBMS TYPE=ROUTE (see "Disposition and Message Routing").

SAVE
specifies that the user-supplied data area (address at TCTTEDA or TCAMSIOA) is to be saved.

ERASE
specifies that a 3270 buffer or 3604 screen is to be erased before this page of output is displayed. The 3284/3286 buffer may contain meaningless data if all positions are not filled with current data.

LAST
signals CICS/VS that this is the last output for a transaction and, therefore, the end of a bracket operation. This operand is meaningful only for VTAM-supported terminals and is applicable only when OUT is the specified disposition.

IOTYPE=
specifies when the output operation is to be started. This operand is meaningful only for VTAM-supported terminals and is applicable only when OUT is the specified disposition.

IMMED
causes the output operation to be started immediately. In cases where WAIT may be coded separately from DFHBMS TYPE=TEXTBLD, overlap of terminal I/O operations is possible by using IMMED and coding the WAIT separately.

DELAY
can cause the output operation to be started when TCP is next dispatched.

If this operand is omitted, IOTYPE defaults to the transaction option specified by TIOTYPE in the DFHPCT TYPE=ENTRY macro instruction. A task using deferred write or message protection always defaults to DELAY regardless of the PCT specification.

LDC=
specifies the mnemonic to be used by CICS/VS to determine the logical device code that is to be used for the TEXTBLD operation and transmitted in the function management header to the logical unit. This operand is meaningful only for VTAM terminals with LDC support.

mnemonic
is the two-character mnemonic used to determine the appropriate LDC numeric value. The mnemonic represents a LDC entry in the DFHTCT TYPE=LDC macro instruction.

YES
indicates that the application program has placed the LDC mnemonic in TCAMSLDM.

If this operand is omitted for a VTAM-supported terminal, a default LDC is chosen as described for the DFHBMS TYPE=PAGEBLD macro instruction.

HEADER=
specifies that header data is to be placed at the beginning of each output page and points to that data (see programming note 2).

symbolic address
is the symbolic address of the header record that will be used to place header information at the beginning of each page.

YES
indicates that the application programmer has placed the address of the header record in TCAMSHDR prior to issuing this DFHBMS macro instruction.

If this parameter is used in a DOS COBOL program the label must not be longer than eight characters.

TRAILER=
specifies that trailer data is to be placed at the bottom of each page and points to that data (see programming note 3).

symbolic address
is the symbolic address of the trailer record that will be used to place trailer data at the bottom of each page.

If this parameter is used in a DOS COBOL program the label must not be longer than eight characters.

YES
indicates that the application programmer has placed the address of the trailer record in TCAMSTRL prior to issuing this DFHBMS macro instruction.

JUSTIFY=
describes the positioning of the text data.

FIRST
indicates that this TEXTBLD data is to be positioned at the top of the page. Any partially formatted page from preceding DFHBMS requests is considered to be complete. If the HEADER operand is specified, the header precedes the TEXTBLD data.

LAST
indicates that this TEXTBLD data is to be positioned at the bottom of the page. If the TRAILER operand is specified, the trailer appears after the TEXTBLD data. The page is considered to be complete after the request is processed.

nnn
indicates that this TEXTBLD data is to be positioned at line nnn of the page.

YES
indicates that the application programmer has placed a binary value from 1 to 255 in TCAMSJ prior to issuing this DFHBMS TYPE=TEXTBLD macro instruction. A value in the range from 1 through 240 represents a line number; 254 represents FIRST; and 255 represents LAST. The values from 241 through 253 are reserved and should not be specified.

CTRL=
is used to specify device characteristics related to terminals of the 3270 Information Display System.

PRINT
must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed. This operand is ignored for 3275s without the Printer Adapter feature and for 3277s.

L40,L64,L80,HONEOM
are mutually exclusive options that control the line length on the printer. L40, L64, and L80 force a carrier return/line feed after 40, 64, or 80 characters, respectively. HONEOM causes the printer to honor all new-line (NL) characters and the first end-of-message (EM) character that appear in displayable fields of the data stream. If the latter option is specified, the application program must insert the NL and EM characters into the data stream. If the NL character is omitted, a carrier return/line feed occurs at the physical end of the carriage. If the EM character is omitted, printing stops at the end of the 3270 buffer.

FREEKB
specifies that the keyboard should be unlocked after this data is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM
activates the 3270 audible alarm feature. For VTAM terminals supporting functional map headers (FMH), ALARM signals BMS to set the alarm flag in the FMH.

PROPT=NLEOM
is applicable to output to 3284, 3286, and 3275 with the printer adapter feature. This parameter is restricted to assembler language programs. If it is to be used, it must be coded with the first DFHBMS request for a logical message.

When this parameter is used, data for these terminals is built with new line characters, as with any other hard copy device. An end-of-message character (EOM) is placed at the end of the data. As the data is printed a new line character in the data terminates printing of a line, and the EOM character terminates printing of the data.

If this parameter is not used and one of the CTRL parameters L40, L64, L80, or NONEOM is not used, the data is printed without regard to the presence of new line or EOM characters in the data; the entire buffer is printed in lines the width of the printer.

The following restrictions apply when using this parameter: Buffer updating and attribute modification of fields previously written into the buffer are not allowed. BMS issues an ERASE with every write to the terminal.

The new line character occupies a buffer position. A number of buffer positions, equivalent to the value of the PGESIZE parameter of the DFHTCT macro for that terminal, is unavailable for data. This may cause data to wrap around in the buffer; if this occurs, it is necessary to reduce the page size specifications in the TCT for the terminal.

CURSOR=
is used to position the cursor upon completion of a write operation to a 3270 device.

number
is an integer indicating a particular position on the screen relative to zero the range of values that may be specified depends upon the screen size of the 3270 being used.

YES
indicates that a value indicating the desired cursor position has been placed in TCABMSCP.

REQID=
specifies the prefix to be used with the Temporary Storage identification. The identification (including the prefix) is used by CICS/VS when attempting message recovery.

BMS message recovery is provided for a logical message only if the STORE operand is specified in the BMS output request and if the logical end of task has been reached.

Only one prefix can be specified for each logical message. If the REQID operand is not specified, CICS/VS assigns the prefix ** (two asterisks).

'prefix'
indicates the alphanumeric prefix to be used as the first two characters of a Temporary Storage identification.

YES
indicates that the prefix has been stored at TCAMSRID.

FMHPARM=
specifies information to be included in a function management header (FMH) being transmitted to a 3650 logical unit. Refer to the CICS/VS Advanced Communication Guide for details on the FMH and on 3650 logical units.

This operand applies only to 3650 logical units with outboard formatting. It specifies the name of the map to be used with this BMS request.

parameter
specifies the eight-character name of the map.

YES
indicates that the map name has been stored in the eight-character TCAMSFMP field.

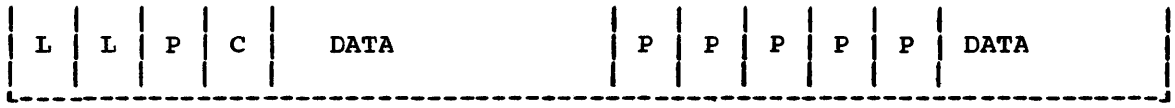
WRBRK=symbolic address
specifies the symbolic address of the routine to receive control when the ATTN key on a 2741 is pressed during the actual write to the terminal. This operand is operative when 2741 Write Break support has been generated into CICS/VS (available only under OS/VS) and when the task would have normally regained control. It is not operative when TYPE=STORE or TYPE=RETURN is specified.

NORESP, TSIOERR, INVREQ, INVLDC, RETPAGE, IGREQID, and ERROR
are used to test the response of BMS to this request for services. These operands can be specified in this macro instruction or in a DFHBMS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for BMS Services."

Programming Notes:

1. See programming note 1 under "Cumulative Page Building with Mapping," above.

2. The data area pointed to by the HEADER operand has the following format:



where:

LL
is a two-byte field containing the length of the data.

P
is a one-byte field indicating the content of any byte in the user data area that BMS is to recognize as a page number locator byte. The following values are reserved and cannot be used: X'0C', X'15', X'17', and X'26'. If automatic page numbering is not desired, this byte must contain a blank.

C
is a reserved one-byte field.

PPPPP
is up to five page number locator bytes, each of which contains the value specified for a page number locator byte. Page numbering up to 32,767 is possible. Consecutive page numbers are inserted automatically by BMS on each page of output. A DFHBMS TYPE=PAGEOUT request or a change in disposition causes the page number to be reset to 1.

DATA
is header information (constant data) to be placed at the beginning of each page of output. Embedded new-line characters (X'15') may be used to provide multiple heading lines.

3. The format of trailer information is the same as the format of header information, described above. Page numbering can be accomplished automatically as with header data.

Noncumulative Page Building

An output request in which neither TEXTBLD nor PAGEBLD is specified can be issued by the application program. Such a request may cause multiple pages to be written as output, but multiple requests cannot be issued to accumulate and format data within one page. One map may be used to format data on one page, and, as in pre-VS BMS, that page may be written directly to the terminal (TYPE=OUT). The rules governing this type of output are as follows:

- Multiple requests cannot be accumulated to build one page, whether mapped or unmapped.
- When using maps, one request cannot build more than one page.
- When not using maps, a single request can result in more than one page.
- If the disposition is STORE, multiple requests can cause multiple pages (each request starting a new page) to be included in one logical message.
- For both mapping and nonmapped operations, if the disposition is STORE, a DFHBMS TYPE=PAGEOUT request must be issued to terminate the logical message.

```
DFHBMS  TYPE=([{OUT[,WAIT]}]  [,NOEDIT][,SAVE][ERASE][,LAST])
               [{STORE    }]
               [{RETURN   }]
        [,IOTYPE={IMMED}]
                 {DELAY}
        [,LDC={mnemonic}]
              {YES     }
        [,DATA={NO  }]
               {YES }
               {CNLY}
        [,MAP={map name}]|[{,MAPADR= symbolic address}]
              {YES     }  [{          YES           }]
        [,MAPSET={mapset name}]|[{,MSETADR= symbolic address}]
                 {YES        }  [{             YES          }]
        [,CTRL=([PRINT][,{L40 }][,FREEKB][,ALARM][,FRSET])]
                         {L64 }
                         {L80 }
                         {HONEOM}
        [,PROPT=NLECM]
        [,CURSOR={number}]
                 {YES   }
        [,FMHPARM={parameter}]
                  {YES      }
        [,WRBRK=symbolic address]  CICS/OS/VS only
        [,NORESP=symbolic address]
        [,TSIOERR=symbolic address]
        [,INVREQ=symbolic address]
        [,INVLDC=symbolic address]
        [,RETPAGE=symbolic address]
        [,INVMPSZ=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbolic address]
```

where:

TYPE=
indicates the general output functions required.

Disposition

OUT
indicates that the output is to be written to the originating terminal at once if that terminal is to receive it.

The application program may issue DFHSC TYPE=FREEMAIN, RELEASE=ALL requests when using DFHBMS TYPE=OUT only under the following conditions:

- Prior to any DFHBMS requests
- Between a DFHBMS TYPE=PAGEOUT and a subsequent BMS request.
- After a DFHBMS TYPE=PURGE request

WAIT
indicates that BMS is to wait until all output operations are complete before returning control to the application program. WAIT must be specified with every output request except the following:

- the last output request prior to task termination
- the last output request prior to an input operation
- the last output request prior to issuing a DFHBMS TYPE=PAGEOUT macro instruction that precedes task termination or an input operation

STORE
indicates that the output is to be placed in temporary storage to be displayed in response to paging commands entered by the terminal operator (for more information about these commands, see the CICS/VS Terminal Operator's Guide). If STORE is specified, with a REQID that is defined in the Temporary Storage Table (TST), CICS/VS provides message recovery for logical messages if the task has reached logical end.

RETURN
indicates that the complete page(s) is to be returned to the application programmer (see programming note 1 under "Cumulative Page Building with Mapping"). The application program regains control (1) immediately following the BMS instruction, or (2) at an alternative entry point specified through the RETPAGE operand of this macro instruction.

If no disposition is specified, the output is sent to the originating terminal. Once the disposition has been established for a logical message, it is not necessary to repeat the disposition for that logical message. Any change of disposition specified while in the process of building a logical message forces that logical message to completion with its original disposition. Then a new logical message is started with the new disposition. The disposition parameter is handled differently under DFHEMS TYPE=ROUTE (see "Disposition and Message Routing").

NOEDIT
specifies that CICS/VS need not insert device-dependent control characters (carrier return, line feed, idle characters, and so on) into the output data stream. The application program, therefore, assumes responsibility for providing any required control characters. This parameter is ignored for all output operations specifying maps.

SAVE
specifies that the user-supplied data area (address at TCTTEDA or TCAMSIOA) is to be saved.

ERASE
specifies that a 3270 buffer or 3604 screen is to be erased before this page of output is displayed. The 3284/3286 buffer would contain meaningless data from prior messages if all positions are not filled with current data.

LAST
signals CICS/VS that this is the last output for a transaction and, therefore, the end of a bracket operation. This operand is meaningful only for BMS-supported VTAM terminals and is applicable only when OUT is the specified disposition.

IOTYPE=

specifies when the output operation is to be started. This operand is meaningful only for VTAM-supported terminals and is applicable only when OUT is the specified disposition.

IMMED

causes the output operation to be started immediately. In cases during which WAIT may be coded in a separate DFHBMS

macro instruction, overlap of terminal I/O operations is possible by using IMMED and coding the WAIT separately.

DELAY
can cause the output operation to be started when TCP is next dispatched.

If this operand is omitted, IOTYPE defaults to the transaction option specified by TIOTYPE in the DFHPCT TYPE=ENTRY macro instruction. A task using deferred write or message protection will override this parameter and not be done until sync point (DFHSP) or task termination.

LDC=

specifies the mnemonic to be used by CICS/VS to determine the logical device code that is to be used for the operation and transmitted in the function management header to the logical unit. This operand is meaningful only for VTAM-supported terminals with LDC support.

mnemonic
is the two-character mnemonic used to determine the appropriate LDC numeric value. The mnemonic represents a LDC entry in the DFHTCT TYPE=LDC macro instruction.

YES
indicates that the application program has placed the LDC mnemonic in TCAMSLDM.

If this operand is omitted for a VTAM-supported terminal, a default LDC is chosen (see the description of the DFHBMS TYPE=PAGEBLD macro instruction).

DATA=

indicates one of three output mapping data selection modes.

NO
specifies that only default data is to be written from the selected map.

YES
indicates that default data from the selected map is to be merged with data placed in the TIOA by the application program.

ONLY
specifies that only application-program-supplied data is to be written.

This operand is valid only when mapping is used. If it is omitted, DATA=NO is assumed.

MAP=

specifies the name of the map to be used when mapping formatted pages.

map name
is the one- to seven-character name of the map within a map set.

YES
indicates that the application programmer has placed the name of the map in TCABMSMN prior to issuing this DFHBMS macro instruction. The name must be left-justified and padded with trailing blanks to eight characters.

If no map set (MAPSET or MSETADR operand) is specified in this DFHBMS macro instruction, the specified map name is taken as the name of the map set.

MAPADR=
specifies the address of the map to be used when mapping formatted pages.

symbolic address
is the one- to seven-character symbolic label that has been assigned to the map coded within the Assembler-language application program.

YES
indicates that the application programmer has placed the address of the map in TCABMSMA prior to issuing the DFHBMS macro instruction.

MAPSET=
specifies the name of the map set to be used in the mapping operation.

map set name
is the one- to seven-character name of the map set.

YES
indicates that the application programmer has placed the name of the map set in TCAMSMSN prior to issuing the DFHBMS macro instruction. The name must be left-justified and padded with trailing blanks to eight characters.

The map set established by this operand must reside in the CICS/VS program library, and a corresponding entry for the map set must exist in the processing program table (PPT).

IF MAPSET is specified, MAP is required.

MSETADR=
specifies the address of the map set to be used in the mapping operation.

symbolic address
is the one- to eight-character symbolic label that has been assigned to the map set coded within the Assembler-language application program.

YES
indicates that the application programmer has placed the address of the map set in TCAMSMSA prior to issuing this DFHBMS macro instruction.

MAPSET and MSETADR are mutually exclusive operands. If MSETADR is specified, MAP is required.

CTRL=
is used to specify device characteristics related to terminals of the 3270 Information Display System.

PRINT
must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed. This operand is ignored for 3275s without the Printer Adapter feature and for 3277s.

L40, L64, L80, HONECM
are mutually exclusive options that control the line length on the printer. L40, L64, and L80 force a carrier return/line feed after 40, 64, or 80 characters, respectively. HONEOM causes the printer to honor all new-line (NL) characters and the first end-of-message (EM) character that appear in displayable fields of the data stream. If the latter option is specified, the application program must insert the NL and EM characters into the data stream. If the NL character is omitted, a carrier return/line feed occurs at the physical end of the carriage. If the EM character is omitted, printing stops at the end of the 3270 buffer.

FREEKB
specifies that the keyboard should be unlocked after this map is written out. If omitted, the keyboard status remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM
activates the 3270 audible alarm feature. For VTAM terminals supporting functional map headers (FMH), ALARM signals BMS to set the alarm flag in the FMH.

FRSET
is valid only when mapping is used and indicates that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before any map data is written to the buffer. This allows the DFHMDF ATTRB specification for the requested map to control the final status of any fields written or rewritten in response to a DFHBMS macro instruction.

If this operand is specified in the DFHBMS macro instruction, it is taken as a total replacement for any CTRL specification in a DFHMSD macro instruction used to create a referenced map set.

PROPT=NLEOM
is applicable to output to 3284, 3286, and 3275 with the printer adapter feature. This parameter is restricted to assembler language programs. If it is to be used, it must be coded with the first DFHBMS request for a logical message.

When this parameter is used, data for these terminals is built with new line characters, as with any other hard copy device. An end-of-message character (EOM) is placed at the end of the data. As the data is printed a new line character in the data terminates printing of a line, and the EOM character terminates printing of the data.

If this parameter is not used and one of the CTRL parameters L40, L64, L80, or NONEOM is not used, the data is printed without regard to the presence of new line or EOM characters in the data; the entire buffer is printed in lines the width of the printer.

The following restrictions apply when using this parameter: Buffer updating and attribute modification of fields previously written into the buffer are not allowed. BMS issues an ERASE with every write to the terminal.

The new line character occupies a buffer position. A number of buffer positions, equivalent to the value of the PGESIZE parameter of the DFHTCT macro for that terminal, is unavailable for data. This may cause data to wrap around in the buffer; if this occurs, it is necessary to reduce the page size specifications in the TCT for the terminal.

CURSOR=
is used to position the cursor upon completion of a write operation to a 3270 device. This operand is valid in this macro instruction only when maps are used.

number
is an integer indicating a particular position relative to zero on the screen; the range of values that may be specified depends upon the screen size of the 3270 being used.

YES
indicates that a value indicating the desired cursor position has been placed in TCABMSCP.

An alternate method may be used to dynamically position the cursor on the output screen. This method is called symbolic cursor positioning (SCP). SCP allows a field in the TIOA to be marked, symbolically, such that the cursor is placed under the first data byte of the field on the output screen.

Requirements for SCP use are as follows:

- MODE=INOUT must be specified on the DFHMSD macro for maps and DSECTs which will be used with SCP.
- CURSOR=YES must be specified on the DFHBMS macro.
- Field TCABMSCP must be initialized with hexadecimal Fs; for example, MVC TCABMSCP,=X'FFFF'.
- The length field, suffix "L", associated with the field under which the cursor is to be placed must be initialized with hexadecimal Fs. For example, MVC FIELD3L,=X'FFFF'.

The remainder of the TIOA may be built as desired by the user. SCP is operable only for devices which allow cursor placement to be performed independent of data placement; for example, 3604 and 3270. SCP specification is ignored for other devices.

REQID=
specifies the prefix to be used with the Temporary Storage identification. The identification (including the prefix) is used by CICS/VS when attempting message recovery.

BMS message recovery is provided for a logical message only if the STORE operand is specified in the BMS output request and if the logical end of task has been reached.

Only one prefix can be specified for each logical message. If the REQID operand is not specified, CICS/VS assigns the prefix ** (two asterisks).

'prefix'
indicates the alphanumeric prefix to be used as the first two characters of a Temporary Storage identification.

YES
indicates that the prefix has been stored at TCAMSRID.

FMHPARM=
specifies information to be included in a function management header (FMH) being transmitted to a 3650 logical unit. Refer to the CICS/VS Advanced Communication Guide for details on the FMH and on 3650 logical units.

This operand applies only to 3650 logical units with outboard formatting. It specifies the name of the map to be used with this BMS request.

parameter
specifies the eight-character name of the map.

YES
indicates that the map name has been stored in the eight-character TCAMSFMP field.

WRBRK=symbolic address
specifies the symbolic address of the routine to receive control when the ATTN key on a 2741 is pressed during the actual write to the terminal. This operand is operative when 2741 Write Break support has been generated into CICS/VS (available only under OS/VS) and when the task would have normally regained control. It is not operative when TYPE=STORE or TYPE=RETURN is specified.

NORESP, TSIOERR, INVREQ, INVIDC, RETPAGE, INVMPSZ, IGREQID, and ERROR
are used to test the response of BMS to this request for services. These operands can be specified in this macro instruction or in a DFHBMS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for BMS Services."

Terminating a Logical Message

When the combining of pieces of data to form a logical message has been requested by means of DFHBMS TYPE=PAGEBLD or TYPE=TEXTBLD macro instruction(s), such combining must be terminated by means of a DFHBMS TYPE=PAGEOUT macro instruction. A logical message created by means of one or more noncumulative output requests with STORE disposition must be terminated by a DFHBMS TYPE=PAGEOUT macro instruction. The format of this macro instruction is as follows:

```
DFHBMS  TYPE=(PAGEOUT[,LAST])
        [,CTRL=([{PAGE     }][,{RETAIN  }])]
                 [{AUTOPAGE}]  [{RELEASE}]
        [,TRAILER={symbolic address}]
                  {YES             }
        [,TRANSID=transaction code]
        [,WRERK={symbolic address}]
                {CURRENT         }
                {ALL             }
        [,EODPURG={AUTO}]
                  {OFER}
        [,REQID={'prefix'}]
                {YES     }
        [,FMHPARM={parameter}]
                  {YES      }
        [,NORESP=symbolic address]
        [,TSIOERR=symbolic address]
        [,RETPAGE=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbclic address]
```

where:

TYPE=

indicates the output functions to be performed.

PAGEOUT

specifies the termination of a logical message. No data is formatted in respcnse tc this request. Any remaining data in the page buffer is processed according to the OUT, STORE, or RETURN described in the previous macro instruction. If a logical message is being built for a routing environment, PAGEOUT completes the lcgical message under route. An additional PAGEOUT macro instruction is required to complete a logical message to the originating terminal.

If an error occurs during PAGEOUT processing, control is returned to the applicatiCn program, and the RETAIN or RELEASE specifications are ignored. The logical message is not considered complete. The applicatiCn program should either retry the PAGEOUT operation or PURGE the message.

LAST

signals CICS/VS that this is the last output for a transaction and, therefore, the end of a bracket operation. If RELEASE is specified, LAST is assumed, unless the PAGEOUT is terminating a route operation.. This operand is meaningful only fcr VTAM-supported terminals.

CTRL=

specifies how pages are to be displayed at the terminal (when the disposition is OUT or STORE) and whether or not control is to be returned to the application program.

PAGE
: specifies that pages are to be paged one at a time to the terminal. BMS writes the first page to the terminal when the terminal becomes available or upon request of the operator. All subsequent pages are written to the terminal in response to a terminal operator request (see the description of paging commands in the CICS/VS Terminal Operator's Guide). If automatic paging was specified for the terminal at system generation, this specification overrides the automatic paging for this logical message. For VTAM-supported terminals, PAGE applies to all LDC page sets accumulated within the logical message.

AUTOPAGE
: specifies that pages are to be paged automatically to the terminal. BMS writes each page of the logical message to the terminal when it becomes available. If paging upon request was specified for the terminal at system generation, this specification overrides for this logical message, provided that the terminal is not a 3270 video terminal (AUTOPAGE cannot be specified for a 3270 video terminal). For VTAM-supported terminals, AUTOPAGE applies to all LDC page sets accumulated in the logical message.

A specification of PAGE for 3284 or 3286 devices is ignored. That is, AUTOPAGE is assumed for these devices. If neither PAGE nor AUTOPAGE is specified, the paging status specified for the terminal at system generation determines how pages are to be written to the terminal. For VTAM-supported terminals with LDC support, paging status for each LDC is obtained from the system LDC table.

RETAIN
: indicates that BMS is to return control to the application program for further processing after it has written the page(s) to the terminal and has received data other then a purge, copy, or paging command from the operator. (See programming note 4.)

RELEASE
: indicates that control is to be returned to the program at the next higher logical level after BMS has written the page(s) to the terminal. When RELEASE is specified, LAST is assumed for VTAM-supported terminals, except when the PAGEOUT is for a route operation.

If neither RETAIN nor RELEASE is specified, and STORE is the disposition for the logical message, a new task is scheduled by CICS/VS task control for writing the pages to the terminal, and control is returned to the application program at this time rather than after the pages are written. After the application program has terminated, the pages will be written to the terminal in response to terminal operator requests (see the description of paging commands in the CICS/VS Terminal Operator's Guide). If pages are being routed, a specification of either RELEASE or RETAIN is ignored.

To ensure that a logical message appears at the receiving terminal at once, before any messages that may have been routed to it, CTRL=RELEASE should be specified. This is especially

true if this transaction is chained to a previous transaction (see "Message Chaining" in the CICS/VS Terminal Operator's Guide for a discussion of chaining).

TRAILER=
specifies that trailer data is to be placed at the bottom of the last page and points to that data (see programming notes 2 and 3 under "Cumulative Page Building without Mapping").

symbolic address
is the symbolic address of the trailer record that will be used to place trailer data at the bottom of the last page. If this parameter is used in a DOS COBOL program the label must not be longer than eight characters.

YES
indicates that the application programmer has placed the address of the trailer record in TCAMSTRL prior to issuing this DFHBMS macro instruction.

This operand is invalid when the logical message being terminated by this PAGEOUT request was built using PAGEBLD requests. If this is the case, BMS returns an INVREQ return code.

TRANSID=transaction code
specifies a one- to four-character transaction identification to be used with the next input message entered from the terminal to which this task is attached.

This operand is valid only when CTRL=RELEASE is specified.

WRBRK=
is used to specify the action that is to occur if the ATTN key on a 2741 is pressed while data is being written to the terminal.

symbolic address
specifies the symbolic address of the routine to receive control when the ATTN key on a 2741 is pressed during the actual write to the terminal. This operand is operative when 2741 Write Break support has been generated into CICS/VS (available only under OS/VS) and when the task would normally have regained control. It is not valid when CTRL=RELEASE is specified.

CURRENT
specifies that transmission of the current page to the terminal is to cease, but, if autopaging has been requested, transmission of the next page (if any) begins.

ALL
specifies that transmission of the current page to the terminal is to cease and that no additional pages are to be transmitted. The logical message is purged.

Both CURRENT and ALL are meaningful only if Write Break support has been generated into CICS/VS (available only under OS/VS), and if TYPE=STORE was specified in preceding DFHBMS requests, or data has been sent to terminals other than the originating terminal. In these cases, data has been placed in temporary storage and is being displayed by a program other than the one associated with the originating terminal. As for the symbolic address parameter, 2741 Write Break support must have been generated into the system.

EODPURG=
specifies the manner in which CICS/VS deletes the current message.

AUTO
specifies that CICS/VS is to delete the message automatically if the operator enters a transaction that is not a paging command. Alternatively, the operator may delete the message with a purge command (see the CICS/VS Terminal Operator's Guide).

OPER
specifies that CICS/VS is not to delete the message until the terminal operator explicitly requests deletion with a purge command.

Note: If temporary storage is reinitialized, all messages are lost, regardless of any other specifications.

REQID=
specifies the prefix to be used with the Temporary Storage identification. The identification (including the prefix) is used by CICS/VS when attempting message recovery.

BMS message recovery is provided for a logical message only if the STORE operand is specified in the BMS output request and if the logical end of task has been reached.

Only one prefix can be specified for each logical message. If the REQID operand is not specified, CICS/VS assigns the prefix ** (two asterisks).

'prefix'
indicates the alphanumeric prefix to be used as the first two characters of a Temporary Storage identification.

YES
indicates that the prefix has been stored at TCAMSRID.

FMHPARM=
specifies information to be included in a function management header (FMH) being transmitted to a 3650 logical unit. Refer to the CICS/VS Advanced Communication Guide for details on the FMH and on 3650 logical units.

This operand applies only to 3650 logical units with outboard formatting. It specifies the name of the map to be used with this BMS request.

parameter
specifies the eight-character name of the map.

YES
indicates that the map name has been stored in the eight-character TCAMSFMP field.

NORESP, TSIOERR, RETPAGE, IGREQID, and ERROR
are used to test the BMS response to this request for BMS services. These operands can be specified in this macro instruction or in a DFHBMS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for BMS Services."

Programming Notes:

1, 2, 3. See programming notes 1, 2, and 3 under "Cumulative Page Building with Mapping" and "Cumulative Page Building without Mapping."

4. RETAIN is intended to be used for a combination of page display from the page file (logical message built using the STORE disposition) and operator data entry. BMS issues a GET to the terminal after writing the appropriate page(s) to the terminal. BMS issues the GET only if the logical message was built with STORE disposition. If the logical message was not built with STORE disposition, BMS returns control to the application program after the last page is written to the terminal, and without issuing a GET to the terminal.

   The operator may enter any page, purge, or copy commands that are valid for the particular message. Any other entered data is passed back to the application program after the current message is purged. The address of the newly acquired TIOA is in TCTTEDA. A chaining command is not valid at this point because it requests the creation of a new task for the terminal to which a task is already attached.

ABNORMALLY TERMINATING A LOGICAL MESSAGE

To discontinue the process of building a logical message, a DFHBMS TYPE=PURGE macro instruction is issued. This instruction causes the portions of the message already built in main storage or on temporary storage to be deleted and returns control to the application program at the instruction following the DFHBMS TYPE=PURGE macro expansion. (The TYPE=PURGE instruction is not to be used if TYPE=RETURN was used in the BMS PAGEBLD or TEXTBLD request.) The format of the macro instruction is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────────┐
│      │       │                                                      │
│      │DFHBMS │  TYPE=PURGE                                          │
│      │       │                                                      │
└──────┴───────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=PURGE
: specifies that all data prepared for a logical message but not yet transmitted to a terminal is to be purged from the system.

BMS MESSAGE RECOVERY

BMS provides message recovery for routed and non-routed messages. These recoverable messages must satisfy the following requirements:

- The DFHBMS TYPE=STORE operand must have been specified on the BMS output requests that built the logical message.
- The BMS default REQID (**) or the specified REQID for the logical message must have been identified to Temporary Storage Program (via the TST) as recoverable.
- The task that built the message must have reached its logical end of task.
- The Temporary Storage Program (TSP) and the Interval Control Program (ICP) must also support recovery.

MESSAGE ROUTING

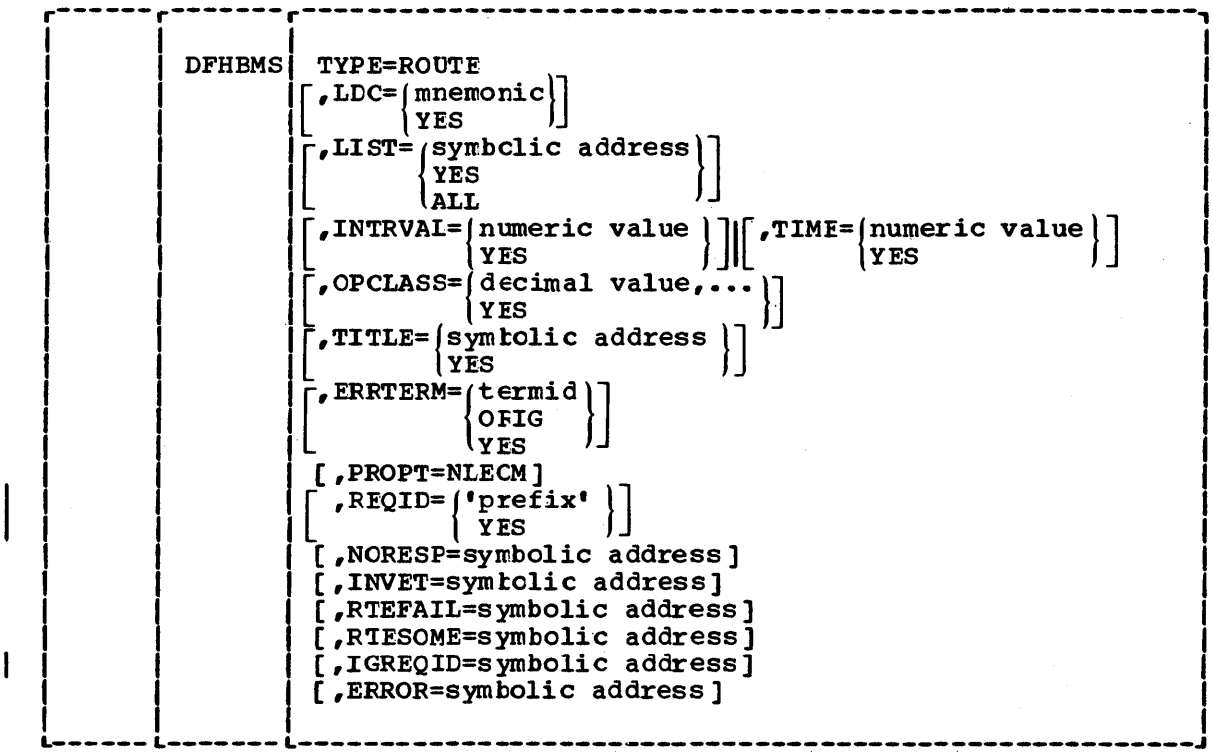
A DFHBMS TYPE=ROUTE request defines the terminal and/or operator environment to receive the message created by subsequent DFHBMS output requests. Routing implies definition of the environment to receive a logical message, but it does not imply transmission of the message. The message may be directed to any or all BMS-supported terminals. A DFHBMS TYPE=PAGEOUT request causes the logical message to be completed and terminates the effect of this DFHBMS TYPE=ROUTE macro instruction.

If a ROUTE request followed by one or more BMS output requests is not terminated by a PAGEOUT request before a subsequent ROUTE request is issued or before the application program terminates, the message is forced to completion. Since the application program did not issue the PAGEOUT request, BMS applies the PAGEOUT defaults to the message. A ROUTE request may be issued immediately following another ROUTE request. In this case, the first ROUTE request is nullified, and the second one determines the routing environment.

A message is considered undeliverable to a destination if it cannot be delivered within a certain interval after the requested delivery time. This interval is specified in the PRGDLAY operand of the DFHSG

PROGRAM=BMS macro instruction by the system programmer. If the PRGDLAY operand is not included, no action is taken for undelivered messages and the message awaits delivery indefinitely. If PRGDLAY is specified, the transient data destination CSMT is notified of the number of undeliverable messages purged for a destination; the application programmer can ensure that additional documentation is provided for an undeliverable message by including the ERRTERM operand (see below) in the DFHBMS TYPE=ROUTE macro instruction. Examples of situations causing undeliverable messages might occur, for example, when a message is routed to a terminal that is out of service, or when an operator identification is specified with a terminal identification and that operator is not signed on that terminal at the time the message is to be delivered.

The format of the DFHBMS TYPE=ROUTE macro instruction is as follows:



where:

TYPE=ROUTE
specifies the initiation of an output page routing operation.

LDC=
specifies the mnemonic to be used by CICS/VS to determine the logical device code that is to be used for the ROUTE operation and transmitted in the function management header (FMH) to the logical unit. The mnemonic provided by the LDC= operand overrides any LDC mnemonic specified in a route list (LIST=symbolic address or YES). This operand is meaningful only for VTAM-supported terminals with LDC support.

mnemonic
is the two-character mnemonic used to determine the appropriate LDC numeric value. The mnemonic represents a LDC entry in the DFHTCT TYPE=LDC macro instruction.

YES
indicates that the application program has placed the LDC mnemonic in TCAMSLDC.

If this operand is omitted and a route list is specified (LIST=symbolic address or YES), the LDC mnemonic in the route list entry is used. If the route list entry contains no LDC mnemonic or no route list is specified, a default LDC is chosen as described for the DFHBMS TYPE=PAGEBLD macro instruction.

LIST=
specifies the terminals and/or operators to which paged data is to be directed.

symbolic address
is the label of a list of terminals and/or operators to which data is to be directed (see programming note 1). If this parameter is used on a DOS COBOL program the label must not be longer than eight characters.

YES
indicates that the address of the list of terminals and/or operators to which data is to be directed has been placed in TCAMSRLA prior to issuing the DFHBMS TYPE=ROUTE macro instruction.

ALL
indicates that all terminals supported by BMS are to receive the paged data.

INTRVAL=
specifies the interval of time after which data being routed to the page file is to be transmitted to the terminal(s).

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59. The minimum value that can be specified is one second.

YES
indicates that the interval of time has been placed in packed decimal form (0HHMMSS+) in TCAMSRTI prior to issuing the DFHBMS TYPE=ROUTE macro instruction.

TIME=
specifies the time of day at which data being routed to the page file is to be transmitted to the terminal(s).

numeric value
is of the form HHMMSS, where HH represents hours from 00 to 99, MM represents minutes from 00 to 59, and SS represents seconds from 00 to 59.

YES
indicates that the time of day has been placed in packed decimal form (0HHMMSS+) in TCAMSRTI prior to issuing the DFHBMS TYPE=ROUTE macro instruction.

OPCLASS=
specifies the operator class(es) to which data is to be routed (see programming note 2).

decimal value,...
consists of one or more decimal values in the range from 1 through 24, separated by commas, specifically identifying the operator class(es).

YES
indicates that values identifying operator classes have been placed in TCAMSOC (three-byte field) prior to issuing the DFHBMS TYPE=ROUTE macro instruction.

Note: A bit position corresponding to each value from 1 through 24 is established in a three-byte field which is matched against the three-byte operator class field in the CICS/VS terminal control table terminal entry (TCTTEOCL) for a terminal. At least one pair of corresponding bits must match in order for the message to be routed to the terminal. The value in TCTTEOCL

is set during sign-on according to the OPCLASS operand of the DFHSNT TYPE=ENTRY macro instruction specified by the system programmer.

TITLE=

specifies the symbolic address of a record that contains a title to be associated with the logical message created under this routing environment (see programming note 3).

symbolic address
: is the symbolic address of the title length field that precedes the title in the title record. If this parameter is used in a DOS COBOL program the label must not be longer than eight characters.

YES
: indicates that the address of the title length field in the title record has been placed in TCAMSTA prior to issuing the DFHBMS TYPE=ROUTE macro instruction.

ERRTERM=

indicates the terminal to be notified if the message is purged because it is undeliverable. The message number, title identification, and destination of the message are indicated.

termid
: is the terminal identification of the terminal to be notified.

ORIG
: indicates that the originating terminal is to be notified.

YES
: indicates that the terminal identification of the terminal to be notified has been placed in TCAMSTI prior to issuing the DFHBMS TYPE=ROUTE macro instruction.

This operand is operative only if the PRGDLAY operand was specified in the DFHSG PROGRAM=BMS macro instruction by the system programmer. If PRGDLAY was not specified, this operand has no effect.

PROPT=NLEOM

is applicable to output to 3284, 3286, and 3275 with the printer adapter feature. This parameter is restricted to assembler language programs. If it is to be used, it must be coded with the first DFHBMS request for a logical message.

When this parameter is used, data for these terminals is built with new line characters, as with any other hard copy device. An end-of-message character (EON) is placed at the end of the data. As the data is printed a new line character in the data terminates printing of a line, and the EOM character terminates printing of the data.

If this parameter is not used and one of the CTRL parameters L40, L64, L80, or NCNEOM is not used, the data is printed without regard to the presence of new line or EOM characters in the data; the entire buffer is printed in lines the width of the printer.

The following restrictions apply when using this parameter: Buffer updating and attribute modification of fields previously written into the buffer are not allowed. BMS issues an ERASE with every write to the terminal.

The new line character occupies a buffer position. A number of buffer positions, equivalent to the value of the PGESIZE parameter of the DFHTCT macro for that terminal, is unavailable for data. This may cause data to wrap around in the buffer; If this occurs, it is necessary to reduce the page size specifications in the TCT for the terminal.

REQID=

specifies the prefix to be used with the Temporary Storage identification. The identification (including the prefix) is used by CICS/VS when attempting message recovery.

BMS message recovery is provided for a logical message only if the STORE operand is specified in the BMS output request and if the logical end of task has been reached.

Only one prefix can be specified for each logical message. If the REQID operand is not specified, CICS/VS assigns the prefix ** (two asterisks).

'prefix'
indicates the alphanumeric prefix to be used as the first two characters of a Temporary Storage identification.

YES
indicates that the prefix has been stored at TCAMSRID.

NORESP, INVET, RTEFAIL, RTESCME, IGREQID, and ERROR
are used to test the BMS response to this request for BMS services. These operands can be specified in this macro instruction or in a DFHBMS TYPE=CHECK macro instruction. The meaning of each operand is discussed in detail under "Test Response to a Request for BMS Services."

Programming Notes:

1. The list of destination terminals and/or operators consists of 16-byte entries as follows:

Bytes	Contents
1-4	contain a four-character (including trailing blanks) terminal or logical unit identification, or blanks
5-6	contain the two-character LDC mnemonic for VTAM-supported terminals with LDC support, or blanks
7-9	contain the operator identification, or blanks
10	serve as a status flag for the route entry (see status flag byte in user-supplied route list)
11-16	reserved; must contain blanks

The end of the list is designated as follows:

```
Assembler:      DC   AL2(-1)
ANS COBOL:      COMPUTATIONAL S9(4) VALUE -1.
PL/I:           DECLARE FIXED BINARY (15) INITIAL (-1);
```

It may be necessary for the application program to supply this list of destinations in noncontiguous areas called segments.

If the list is supplied in segments, every segment except the last is terminated with (at least) an eight-byte entry as fcllows:

Bytes	Contents
1-2	
Assembler:	DC AL2(-2)
ANS COBOL:	COMPUTATIONAL S9(4) VALUE -2.
PL/I:	DECLARE FIXED BINARY (15) INITIAL (-2);
3-4	reserved
5-8	contain the chain address to the first entry of the next segment

The end of the list is designated as described above for an unsegmented list.

If, for any entry in the list,

a. The terminal identification is specified but the operator identification is omitted, the data is routed to that terminal without regard to operator identification.

b. The operator identification is specified but no terminal identification is given, the data is routed to the 'first' terminal at which the operator is signed on under the specified operator identification. The 'first' is determined by the physical location of the terminal entry in the CICS/VS terminal control table. If no operator is signed on under the specified operator identification when the DFHBMS TYPE=ROUTE macro instruction is executed, the route list entry is ignored.

c. Both terminal identification and operator identification are specified, the data is routed to that terminal.

For either b or c above, the data is displayed only if the operator with the specified identification is signed on at the terminal when the data is ready to be displayed, or when the operator signs on after the data is ready to be displayed. Entries of all three types may be included in one segmented or unsegmented list.

It should be noted that the status flag in each route list entry is used to notify the application program of certain status conditions for that requested destination. Therefore, if the route list is contained within the application program and BMS alters the status flag, the application program can no longer be considered reentrant.

2. If data is to be routed tc an operator class, the application programmer may do one of the following:

   a. Specify OPCLASS and omit LIST. Data is routed to each terminal at which an operator is signed on with the specified OPCLASS at the time the DFHBMS macro instruction is issued.

   b. Specify OPCLASS and LIST=ALL. Data is routed to all terminals.

   In both cases, the data is not displayed on a terminal until an operator is signed on with the specified OPCLASS. In general, LIST=ALL is specified with OPCLASS only when it is anticipated that someone will eventually sign on with the specified OPCLASS at every supported terminal.

   If the application programmer specifies OPCLASS and LIST=symbolic address, and the list contains operator identifications, a specified operator identification overrides OPCLASS for that entry.

3. The title pointed to by the TITLE operand is displayed with the logical message ID when the terminal paging query command is entered (see the CICS/VS Terminal Operator's Guide). This title serves as an additional message identifier, displayed upon request with the message ID, not on the logical message. The value in the two-byte length field preceding the title includes the bytes used for the length field. The length field and title, in total, may be up to 64 bytes long. For example:

```
|X'001A'|MONTHLYØINVENTORYØREPORT|
└-------------------------------┘

  2-byte          24-byte
  length        title field
  field
```

Disposition and Message Routing

A routed logical message can be built using either of two dispositions: STORE or RETURN. The first BMS output request issued following the ROUTE request (with some exceptions noted below) determines the disposition of the logical message. This first request may specify STORE or RETURN; if neither is specified, the default is STORE. Once established, the disposition remains unchanged until the logical message is completed (PAGEOUT). It need not be repeated for subsequent requests. An output request specifying a disposition that is not in effect results in a return code of INVREQ.

A disposition of STORE is the normal disposition and finally results in the message either being delivered or purged. A disposition of RETURN causes the routed logical message to be returned to the application program. It is the responsibility of the application program to deliver the logical message.

A task can converse with the terminal to which it is currently attached (assuming the task is terminal-oriented) during the time that it is building the logical message. That attached terminal is known as the direct terminal; a terminal to which the message is to be routed is known as a routing terminal. If any input requests (DFHBMS TYPE=IN or TYPE=MAP) are encountered while the message is being built, they are processed as usual. To transmit output to the direct terminal while the routed logical message is being built, the task can issue non-TEXTBLD, non-PAGEBLD requests with an explicit disposition of OUT.

The disposition of OUT isolates the output request to the direct terminal from the requests that are building the routed logical message.

The following points summarize rules for conversation with the direct terminal while a routed logical message is being built:

- OUT must be specified in any output request that is to go to the direct terminal.
- TEXTBLD and PAGEBLD requests with a disposition of OUT are invalid and result in a return code of INVREQ.
- The direct terminal may be included in the routing environment without impairing the ability to converse with it while under ROUTE. Data routed to the direct terminal will be delivered as though the ROUTE had been issued from another terminal.

A list of "abridged" requests, in order of execution, is given below. The action taken by BMS for each is indicated.

Request	Action Taken by BMS
DFHBMS TYPE=OUT	Transmit to direct terminal.
DFHBMS TYPE=ROUTE	Establish routing environment.
DFHBMS TYPE=OUT	Transmit to direct terminal.
DFHBMS TYPE=IN	Receive from direct terminal.
DFHBMS TYPE=TEXTBLD	First output request eligible for routing establishes default disposition of STORE and TEXTBLD as mode of page building.
DFHBMS TYPE=OUT	Transmit to direct terminal.
DFHBMS TYPE=TEXTBLD,RETURN	INVREQ - routed logical message has already established a disposition of STORE.
DFHBMS TYPE=TEXTBLD	Continue building routed logical message.
DFHBMS TYPE=PAGEBLD,STORE	INVREQ - routed logical message being built with TEXTBLD requests cannot tolerate PAGEBLD request.
DFHBMS TYPE=PAGEBLD,OUT	INVREQ - cannot issue PAGEBLD or TEXTBLD request to direct terminal while building a routed logical message.
DFHBMS TYPE=TEXTBLD,STORE	Continue building routed logical message.
DFHBMS TYPE=PAGEOUT	Terminate routed logical message and routing operation.
DFHBMS TYPE=OUT	Transmit to direct terminal.

TEST RESPONSE TO A REQUEST FOR BMS SERVICES

When issuing a request for BMS services, the application programmer can check the response of BMS to determine subsequent processing that should be carried out. One step in doing so is to specify the entry-point names (symbolic addresses) of user-written

exception-handling routines, any of which may be executed as a result of the check. The addresses can be specified in any of three ways:

1. Include the entry-point names in operands of the DFHBMS macro instruction by which the BMS service is requested.

2. Include the entry-point names in operands of a DFHBMS TYPE=CHECK macro instruction immediately following the DFHBMS macro instruction by which the BMS service is requested.

3. Include instructions immediately following the DFHBMS macro instruction that examine the response codes set automatically by BMS when making the response, and transfer control to an appropriate user-written exception-handling routine accordingly.

Under either of the first two methods above, CICS/VS checks the response codes that it sets and transfers control to the exception-handling routine named in the operand associated with the condition that has occurred (if that operand has been specified). The application programmer need not be concerned with which response code corresponds to which condition. It is only necessary to understand the keyword operands and provide for all conditions that may occur.

The keyword operands that are applicable in various forms of the DFHBMS macro instruction are noted in the explanations of the macro instruction formats earlier in this chapter. The format of the DFHBMS TYPE=CHECK macro instruction is as follows:

```
┌──────┬───────┬──────────────────────────────────────────────────┐
│      │       │                                                  │
│      │DFHBMS │ TYPE=CHECK                                       │
│      │       │ [,NCRESP=symbolic address]                       │
│      │       │ [,TSIOERR=symbolic address]                      │
│      │       │ [,INVREQ=symbolic address]                       │
│      │       │ [,RETPAGE=symbolic address]                      │
│      │       │ [,MAPFAIL=symbolic address]                      │
│      │       │ [,INVET=symbolic address]                        │
│      │       │ [,INVLDC=symbolic address]                       │
│      │       │ [,RTEFAIL=symbolic address]                      │
│      │       │ [,RTESOME=symbolic address]                      │
│      │       │ [,INVMPSZ=symbolic address]                      │
│      │       │ [,IGREQID=symbolic address]                      │
│      │       │ [,ERROR=symbolic address]                        │
│      │       │                                                  │
└──────┴───────┴──────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
: indicates that the BMS response to a request for BMS services is to be checked.

NORESP=symbolic address
: specifies the entry label of the user-written routine to which control is passed if none of the other response conditions (whether checked for or not) occurs. NORESP signifies "normal response."

TSIOERR=symbolic address
: specifies the entry label of the user-written routine to which control is to be passed if an unrecoverable temporary storage input/output error occurs.

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is passed if the request for BMS services is invalid. This response may be caused by any of the following conditions:

- Changing the disposition of a routed logical message prior to its completion, through DFHBMS TYPE=PAGEOUT
- Issuing a separate TYPE=TEXTBLD or TYPE=PAGEBLD request to the direct (originating) terminal while in the process of building a routed logical message
- Mixing TYPE=TEXTBLD and TYPE=PAGEBLD requests when building a logical message
- Specifying NOEDIT with a TYPE=PAGEBLD or TYPE=TEXTBLD request
- Specifying the TRAILER operand with TYPE=PAGEOUT when terminating a logical message built using TYPE=PAGEBLD requests
- Issuing a DFHBMS request with DATA=YES or DATA=NO and specifying a map with no field specifications
- Issuing a DFHBMS request with TYPE=STORE from a CICS/VS application program communicating with a host conversational (3653) logical unit.

RETPAGE=symbolic address
specifies the entry label of the user-written routine to which control is passed if one or more completed pages are returned to the application program. This response can occur only if TYPE=RETURN is specified in the DFHBMS macro instruction (see the description of TYPE=RETURN for further information).

MAPFAIL=symbolic address
specifies the entry label of the user-written routine to which control is passed if the data to be mapped has a length of zero or does not contain a SBA (start buffer address) sequence. This response can occur only if TYPE=IN or TYPE=MAP is specified and data is mapped from a 3270 device. For TYPE=IN, the address of the erroneous TIOA is available at TCTTEDA. For TYPE=MAP, this address is wherever the user placed it prior to the request (either in TCTTEDA or TCAMSIOA).

INVET=symbolic address
specifies the entry label of the user-written routine to which control is passed if the terminal identification specified by the ERRTERM operand of a DFHBMS TYPE=ROUTE macro instruction is invalid or is assigned to a terminal of a type not supported under BMS.

INVLDC=symbolic address
specifies the entry label of the user-written routine to which control is passed if the LDC mnemonic specified by the LDC operand does not appear in the LDC list associated with the TCTTE.

RTEFAIL=symbolic address
specifies the entry label of the user-written routine to which control is passed if a DFHBMS TYPE=ROUTE request results in a null routing environment (that is, the message will be sent, by default, to only the originating terminal). (To determine why route list entries were skipped, refer to "Status Flag Byte in User-Supplied Route List.")

RTESOME=symbolic address
specifies the entry label of the user-written routine to which control is passed if (1) some of the entries in the user-specified route list named in the LIST operand of a DFHBMS TYPE=ROUTE macro instruction are excluded from the routing environment, or (2) LIST=ALL is specified and not all of the entries in the terminal control table are included in the routing environment. (To determine why some route list entries were skipped, refer to "Status Flag Byte in User-Supplied Route List.")

INVMPSZ=symbolic address
specifies the entry label of the user-written routine to which control is to be passed if (1) the specified map is too wide for a receiving terminal, or (2) OFLOW has been requested and the specified map is too long for the receiving terminal. Upon entry to the user-written routine, TCAMSRI1 contains a terminal code that further identifies the receiving terminal (see "Terminal Code (TC) Table").

IGREQID=symbolic address
specifies the entry label of a user-coded routine to which control is to be passed if the prefix specified is different from the established (via a previous specification or default) REQID for this logical message.

ERROR=symbolic address
specifies the entry label of the user-written routine to which control is passed if any of the response conditions except NORESP occurs.

Some response codes may appear in combination with other response codes. These combinations are: RTEFAIL and INVET, and RTESOME and INVET. The order used by BMS in checking for all conditions that the application programmer specifies is as follows: NCRESP, TSIOERR, INVREQ, RETPAGE, MAPFAIL, RTEFAIL, RTESOME, INVET, INVMPSZ, and ERROR. Thus, if the application programmer has specified INVET and RTEFAIL and both of these responses apply, BMS transfers control to the user-written exception-handling routine identified in the RTEFAIL operand. In this situation, the INVET operand is not acted upon.

If the application programmer does not provide for the checking for a particular response to a BMS request, and if the exception condition corresponding to that response occurs, program flow proceeds to the instruction following the DFHBMS macro instruction in the application program.

When the third approach identified at the beginning of this discussion is used, the application programmer must know the BMS response codes and their meanings. For this approach, the application programmer can access the response code(s) at TCAMSRC1 and TCAMSRC2. The possible response codes and the conditions to which they correspond are identified in the right-hand columns of Figure 10-3. DFHBMS service requests for which the conditions are applicable are shown at the left. The keywords are explained in detail in the discussion of DFHBMS TYPE=CHECK.

DFHBMS Service Request	Condition	Response Code			Response Code Location
		Assembler	ANS COBOL	PL/I	
INPUT, OUTPUT, ROUTING, CHECK	NORESP (Normal Response)	X'00'	12-0-1-8-9	00000000	TCAMSRC1 and TCAMSRC2
OUTPUT, CHECK	INVREQ (Invalid Request)	X'01'	12-1-9	00000001	TCAMSRC1
OUTPUT, CHECK	RETPAGE (Return Page)	X'02'	12-2-9	00000010	TCAMSRC1
INPUT, CHECK	MAPFAIL (Mapping Attempt Failure)	X'04'	12-4-9	00000100	TCAMSRC1
INPUT, OUTPUT, CHECK	INVMPSZ (Invalid Map Size)	X'08'	12-8-9	00001000	TCAMSRC1
ROUTING, CHECK	INVET (Invalid Error Terminal)	X'20'	11-0-1-8-9	00100000	TCAMSRC1
ROUTING, CHECK	RTESOME (Routing to Only Some Terminals)	X'40'	no punches	01000000	TCAMSRC1
ROUTING, CHECK	RTEFAIL (Routing Failure)	X'80'	12-0-1-8	10000000	TCAMSRC1
INPUT, OUTPUT, ROUTING, CHECK	ERROR (Any Response Other Than NORESP)	See Note	See Note	See Note	TCAMSRC1 TCAMSRC2
OUTPUT, CHECK	TSIOERR (Temporary Storage I/O Error)	X'80'	12-0-1-8	10000000	TCAMSRC2

Note: The test for the ERROR response is satisfied by a not equal condition; that is, not X'00', not 12-0-1-8-9, or not 00000000 for Assembler, ANS COBOL, and PL/I, respectively.

Figure 10-3. BMS Response Codes

The following examples show how to examine the response code provided by BMS at TCAMSRC1 and TCAMSRC2 and transfer control to the appropriate user-written routine accordingly.

For Assembler language:

```
        DFHBMS  TYPE=(TEXTBLD,STORE)    BUILD OUTPUT
        CLI     TCAMSRC1,X'00'          ANY UNUSUAL CONDITIONS, TEST 1
        BNE     ERROR                   ..YES, GO TERMINATE THE TASK
        CLI     TCAMSRC2,X'00'          ..NO, ANY UNUSUAL CONDITIONS, TEST 2
        BE      GOOD                    ..NO, GO CONTINUE PROCESSING
ERROR   DS      0H                      YES, TERMINATE THE TASK
        DFHPC   TYPE=ABEND              TERMINATE THE TASK
GOOD    DS      0H
        .
        .
        .
```

For ANS COBOL:

```
       DFHBMS TYPE=(TEXTELD,STORE)      BUILD OUTPUT
       IF TCAMSRC1 NOT = ' ' THEN GO TO ERROR.
       IF TCAMSRC2 = ' ' THEN GO TC GOOD.
ERROR.
       DFHPC TYPE=ABEND                 TERMINATE THE TASK
GOOD.
       .
       .
       .
```

where the value specified within the single quotation marks is an unprintable multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

For PL/I:

```
       DFHBMS TYPE=(TEXTELD,STORE)      BUILD OUTPUT
       IF TCAMSRC1 ¬= '00000000'B THEN GO TO ERROR;
       IF TCAMSRC2 ='00000000'B THEN GO TO GOOD;
ERROR:
       DFHPC TYPE=ABEND                 TERMINATE THE TASK
GOOD:
       .
       .
       .
```

TERMINAL CODE (TC) TABLE

A terminal code table is established within BMS for reference in servicing BMS-supported terminals. There is one entry in this table for each terminal supported under BMS. The terminal codes that appear in the table are given in Figure 10-4. This code appears in the list of completed pages available at TCAMSRLA when the application programmer has specified that pages of output be returned (that is, RETURN is the disposition parameter in the output request). The code is available at TCAMSRI1 when an invalid map size (INVMPSZ) response is returned.

Code Character	Terminal Type
A	CRLP or TRMTYPE=TCAM terminals
B	Magnetic Tape
C	Sequential Disk
D	TWX Model 33/35
E	1050
F	2740 Models 1 and 2 (Without Buffer Receive)
G	2741
H	2740 Model 2 (With Buffer Receive)
I	2770
J	2780
K	3780
L	3270 Model 1
M	3270 Model 2
Q	2980 Models 1 and 2
R	2980 Model 4
U	3601
V	Host Conversational (3653)
W	3650 User Program
X	3650/3270 Host Conversational (3270)

Figure 10-4. BMS Terminal Code Table

STATUS FLAG BYTE IN USER-SUPPLIED ROUTE LIST

Each route list entry contains a status flag byte used by BMS to indicate to the application program the status of the destination at the time the DFHBMS TYPE=ROUTE macro instruction was issued. Upon return, the application program can investigate the status byte for each route list entry and take appropriate action.

The status flag byte settings are shown in Figure 10-5. Their meanings are explained in greater detail in the text that follows.

Condition (See explanation below.)	Status Flag		
	Assembler	ANS COBOL	PL/I
ENTRY SKIPPED	X'80'	12-0-1-8	10000000
INVALID TERMINAL IDENTIFICATION	X'40'	no punches	01000000
TERMINAL NOT SUPPORTED UNDER BMS	X'20'	11-01-8-9	00100000
OPERATOR NOT SIGNED ON	X'10'	12-11-1-8-9	00010000
OPERATOR SIGNED ON UNSUPPORTED TERMINAL	X'08'	12-8-9	00001000
INVALID LDC MNEMONIC	X'04'	12-4-9	00000100

Figure 10-5. BMS Status Flags

ENTRY SKIPPED A route list entry that is flagged as skipped was not included in the resolved routing environment. If an entry has been skipped, another flag indicating why the entry was skipped may be on in the status byte. This second flag could be one of the following:

- INVALID TERMINAL IDENTIFICATION
- TERMINAL NOT SUPPORTED UNDER BMS
- OPERATOR NOT SIGNED ON - only an operator identification was specified in the route list entry and that operator was not signed on any terminal
- OPERATOR SIGNED ON UNSUPPORTED TERMINAL
- INVALID LDC MNEMONIC

If only the ENTRY SKIPPED flag is on, neither a terminal identification nor an operator identification was specified in the route list entry.

INVALID TERMINAL IDENTIFICATION
This flag indicates that the terminal identification specified in the route list entry does not have a corresponding TCTTE in the terminal control table. This entry is also flagged as ENTRY SKIPPED.

TERMINAL NOT SUPPORTED UNDER BMS

This flag indicates that the terminal identification specified in the route list entry is for a terminal type that is not supported under BMS or the terminal table entry indicated that the terminal identification was not eligible for routing. This entry is also flagged as ENTRY SKIPPED.

OPERATOR NOT SIGNED ON

This flag indicates that the specified operator is not signed on. Any one of the following conditions causes this flag to be set:

1. An operator identification was specified with a terminal identification, but the specified operator was not signed on the terminal. This entry is not skipped.

2. An operator identification was specified without a terminal identification, and the operator was not signed on any terminal. This entry is also flagged as ENTRY SKIPPED.

3. The OPCLASS operand was specified with the DFHBMS TYPE=ROUTE macro instruction and a terminal identification was specified in the route list entry, but the operator signed on the terminal did not qualify under OPCLASS. This entry is not skipped.

OPERATOR SIGNED ON UNSUPPORTED TERMINAL

This flag indicates that only an operator identification was specified in the route list entry, and that operator was signed on a terminal not supported by BMS. This entry is also flagged as ENTRY SKIPPED. The unsupported terminal identification is returned in that route list entry at URLTRMID for informational purposes only.

INVALID LDC MNEMONIC

This flag indicates that one of the following conditions occurred:

1. The LDC mnemonic specified in the route list does not appear in the LDC list associated with the TCTTE.

2. The device type generated in the system LDC table for the specified or implied LDC mnemonic is not the same as the device type for the first LDC specified in the route environment.

A symbolic storage definition of the user-supplied route list is available on the source library(s) under the member name DFHURLDS. This symbolic storage definition can be used as an aid in building the route list, and if necessary, in testing the status flag byte for each entry upon return from a DFHBMS TYPE=ROUTE request that refers to a list. The symbolic base register is URLBAR.

STANDARD ATTRIBUTE LIST AND PRINTER CONTROL CHARACTERS (DFHBMSCA)

The application programmer can obtain a set of commonly used 3270 field attributes and printer control characters by copying DFHBMSCA into his program. For American National Standard (ANS) COBOL, this definition must be copied into the Working Storage Section. DFHBMSCA consists of a set of EQU statements in the case of Assembler language,

a set of 01 statements in the case of ANS COBOL, and DECLARE statements defining elementary character variables in the case of PL/I. One possible use for DFHBMSCA is for the purpose of temporarily changing attribute characters in a map.

The field attributes/printer control characters and corresponding symbolic names are listed in Figure 10-6. These attributes cannot be combined by the application programmer in any manner. If any combinations other than those listed are required, the application programmer must either use the ATTRB operand of the DFHMDF macro instruction to obtain the desired combinations or generate new attribute combinations offline.

Symbolic Name	Field Attribute/Printer Control Character
DFHBMPEM	3270 Printer end of message
DFHBMPNL	3270 Printer new-line character
DFHBMASK	Autoskip
DFHBMUNP	Unprotected
DFHBMUNN	Unprotected and numeric
DFHBMPRO	Protected
DFHBMBRY	High intensity
DFHBMDAR	Dark, nonprint
DFHBMFSE	MDT on
DFHBMPRF	Protected and MDT on
DFHBMASF	Autoskip and MDT on
DFHBMASB	Autoskip and high intensity

Figure 10-6. 3270 Field Attributes and Printer Control Characters

STANDARD ATTENTION IDENTIFIER LIST (DFHAID)

To test the method of initiating an incoming READ from the 3270 Information Display System, the application programmer is provided with a set of 3270 attention identifiers (single-character variables called AIDs) that can be used to test the value at TCTTEAID. He can obtain this set of attention identifiers by copying DFHAID into his program. For ANS COBOL, this definition must be copied into the Working Storage Section.

DFHAID consists of a set of EQU statements in the case of Assembler language, a set of 01 statements in the case of ANS COBOL, and DECLARE statements defining elementary character variables in the case of PL/I. The symbolic names for the attention identifiers and the corresponding 3270 functions are given in Figure 10-7.

Symbolic Name	3270 Function
DFHENTER	Enter key
DFHCLEAR	Clear key
DFHPEN	Immediately detectable field
DFHPA1	PA1 key
DFHPA2	PA2 key
DFHPA3	PA3 key
DFHPF1	PF1 key
.	.
.	.
.	.
DFHPF12	PF12 key

Figure 10-7. 3270 Attention Identifiers and Functions

BMS TIOA SPECIFICATION

Depending on the programming language used, the BMS symbolic storage definition of the TIOA must be provided in the application program as shown in the following examples. Note that mapsetname1, mapsetname2, and mapsetname3 in these examples are the names of members that contain the assembly of a BMS symbolic storage definition (TYPE=DSECT).

1. Assembler-language COPY statements.

```
COPY DFHTIOA
COPY mapsetname1
COPY mapsetname2
COPY mapsetname3
```

2. ANS COBOL COPY statements for each symbolic storage definition. Note that mapname1, mapname2, and mapname3 in this example are the names of the first maps in the map sets.

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBILDS.
02 TIOABAR PICTURE 59(8) COMP.
.
.
.
01 DFHCSADS COPY DFHCSADS.
01 DFHTCADS COPY DFHTCADS.
01 DFHTIOA COPY DFHTIOA.
01 mapname1 COPY mapsetname1.
01 mapname2 COPY mapsetname2.
01 mapname3 COPY mapsetname3.
.
.
```

3. PL/I %INCLUDE statements.

```
%INCLUDE DFHTIOA;;
%INCLUDE mapsetname1;
%INCLUDE mapsetname2;
%INCLUDE mapsetname3;
.
.
.
```

In addition to providing the BMS symbolic storage definition for the TIOA, the application programmer must establish addressability for this storage definition. Depending on the programming language used, this is accomplished as follows:

1. Assembler-language ORG statement immediately preceding the symbolic storage definition for each map, starting with the second map. For example:

```
COPY DFHTIOA
COPY mapsetname1
COPY mapsetname2
COPY mapsetname3
.
.
.
```

```
DFHSC TYPE=GETMAIN,                                          *
      NUMBYTE=mapname.E-TIOADBA,                             *
      CLASS=TERMINAL,                                        *
      INITIMG=00
L     TIOABAR,TCASCSA       ESTABLISH TIOA ADDRESSABILITY
.
.
```

Note: BMS appends an E and hyphen to each application programmer selected mapname or T to each selected mapsetname to create a label at the end of the map description or map set description. The Assembler-language programmer can refer to that label as in the example above to request only the amount of storage required.

2. ANS COBOL 02 level statements immediately following the COPY statement for the Linkage Section Base Locator (BLL). These 02 statements must be coded in the same order as the corresponding 01 statements coded previously. For example:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
.
.
    02 TIOABAR PICTURE S9(8) COMPUTATIONAL.
    02 MAPBASE1 PICTURE S9(8) COMPUTATIONAL.
    02 MAPBASE2 PICTURE S9(8) COMPUTATIONAL.
    02 MAPBASE3 PICTURE S9(8) COMPUTATIONAL.
.
.
01 DFHTIOA COPY DFHTIOA.
01 mapname1 COPY mapsetname1.
01 mapname2 COPY mapsetname2.
01 mapname3 COPY mapsetname3.
.
.
PROCEDURE DIVISION.
.
.
DFHSC TYPE=GETMAIN,                                          *
      NUMBYTE=120,                                           *
      CLASS=TERMINAL,                                        *
      INITIMG=00
    MOVE TCASCSA TO TIOABAR.
    ADD 12 TIOABAR GIVING MAPBASE1, MAPBASE2, AND MAPBASE 3.
```

3. PL/I based pointer variable (BMSMAPBR). For example:

```
DCL TIOABAA FIXED BINARY(31,0) BASED(TIOABAB);
.
.
%INCLUDE DFHTIOA;;
%INCLUDE mapsetname1;  /*EACH OF THESE MAPS IS*/
%INCLUDE mapsetname2;  /*BASED ON THE SAME POINTER*/
%INCLUDE mapsetname3;  /*VARIABLE - BMSMAPBR*/
.
.
DFHSC TYPE=GETMAIN,                                          *
      NUMBYTE=120,                                           *
      CLASS=TERMINAL,                                        *
      INITIMG=00
TIOABAR=TCASCSA;
TIOABAB=ADDR(TIOABAR);    /*TIOABAA NOW OVERLAYS TIOABAR*/
TIOABAA=TIOABAA+12;       /*IN EFFECT, ADDS 12 TO TIOABAR*/
BMSMAPBR=TIOABAR;
```

PROGRAMMING CONSIDERATIONS FOR PAGING COMMANDS ON VIDEO DEVICES

The commands used by terminal operators to communicate with CICS/VS BMS are collectively known as terminal paging commands, or simply as paging commands. Their format and use are discussed in detail in the CICS/VS Terminal Operator's Guide.

The application programmer need not recall all details of these commands. However, it should be clearly understood that use of BMS at map definition time and in executable programs can have a significant effect on terminal operator procedures.

Cursor placement is an important consideration in programming for paging commands. Any of the following items can cause a paging command not to be the first data read by CICS/VS and therefore not to be interpreted as a paging command.

- After a print operation on a 3275 Display Station, the cursor is set to position zero. A paging command entered at this location is not recognized unless the last position of the buffer contains an attribute byte or the buffer has been cleared.

- A field sent with DATA=ONLY and no attribute byte in the TIOA is written into the buffer without an attribute byte. If the application programmer places the cursor in this field and the operator keys a paging command beginning at the cursor location, the paging command is not recognized.

  Since the field has no attribute byte, the hardware considers the data to be an extension of the previously defined field. When the operator keys into the middle of the hardware-recognized field and presses the enter key, the field is transmitted from the beginning of the previously defined field. The data at the beginning of the field is examined for a paging command and responded to accordingly.

- Cursor specification in the DFHBMS macro instruction can adversely affect operator action if the cursor is not set at the beginning of a field. Paging commands entered at a cursor location that is not the beginning of a field are not recognized by BMS because data transmission starts at the beginning of the field if the field is not set to nulls X'00'.

EXAMPLES OF THE USE OF BMS

The examples in this section are based on a fairly simple screen exercising problem and are intended to show the results of generating BMS symbolic storage definitions.

In the examples, input and output symbolic storage definitions are illustrated for each of the programming languages supported by CICS/VS. Each of these examples is generated from the screen definition of the first example; only the initial DFHMSD entry is changed. (See Figures 10-8 through 10-17.)

```
SAMPLE DFHMSD TYPE=DSECT,LANG=ASM,MODE=INOUT,TERM=3270,CTRL=FREEKB
MAP    DFHMDI LINE=1,COLUMN=1,SIZE=(4,80)
       DFHMDF POS=0,LENGTH=17,INITIAL='ENTER YOUR NAME--'
NAME   DFHMDF POS=18,LENGTH=18,ATTRB=(IC,UNPROT)
       DFHMDF POS=37,LENGTH=2
       DFHMDF POS=40,LENGTH=17,INITIAL='WHAT IS THE DATE?'
MONTH  DFHMDF POS=58,LENGTH=2,INITIAL='MM',GRPNAME=DATE
DAY    DFHMDF POS=60,LENGTH=2,INITIAL='DD',GRPNAME=DATE
YEAR   DFHMDF POS=62,LENGTH=2,INITIAL='YY',GRPNAME=DATE
       DFHMDF POS=37,LENGTH=15
       DFHMDF POS=80,LENGTH=26,INITIAL='SELECT YOUR FAVORITE COLOR'
BLUE   DFHMDF POS=120,LENGTH=9,ATTRB=DET,INITIAL='?␣BLUE␣␣␣'
RED    DFHMDF POS=131,LENGTH=8,ATTRB=DET,INITIAL='?␣RED␣␣␣'
AMBER  DFHMDF POS=141,LENGTH=10,ATTRB=DET,INITIAL='?␣AMBER␣␣␣'
       DFHMDF POS=160,LENGTH=19,ATTRB=(PROT,ERT),                    *
             INITIAL='NOW HIT A PF KEY...'
ERROR  DFHMDF POS=240,LENGTH=19,ATTRB=DRK,                           *
             INITIAL='SORRY, TRY AGAIN...'
       DFHMSD TYPE=FINAL
```

Figure 10-8. Symbolic Storage Definition Input

```
MAPI    DS      0C .                INPUT MAP ORIGIN
MAPO    DS      0C .                OUTPUT MAP ORIGIN

NAMEL   DS      H   .                 INPUT DATA FIELD LENGTH
NAMEF   DS      0C .                  DATA FIELD FLAG
NAMEA   DS      C .                   DATA FIELD ATTRIBUTE
NAMEI   DS      0CL18               INPUT DATA FIELD
NAMEO   DS      CL18 .              OUTPUT DATA FIELD

*  START NEW DATA GROUP DATE
DATEL   DS      H   .                 INPUT GROUP FIELD LENGTH
DATEF   DS      0C .                  GROUP FIELD FLAG
DATEA   DS      CL1 .                 GROUP FIELD ATTRIBUTE
DATEI   DS      0C .                  INPUT GROUP FIELD ORIGIN
DATEO   DS      0C .                  OUTPUT GROUP FIELD ORIGIN


MONTHI  DS      0CL2 .              INPUT DATA FIELD
MONTHO  DS      CL2 .               OUTPUT DATA FIELD

DAYI    DS      0CL2 .              INPUT DATA FIELD
DAYO    DS      CL2 .               OUTPUT DATA FIELD

YEARI   DS      0CL2 .              INPUT DATA FIELD
YEARO   DS      CL2 .               OUTPUT DATA FIELD

BLUEL   DS      H   .                 INPUT DATA FIELD LENGTH
BLUEF   DS      0C .                  DATA FIELD FLAG
BLUEA   DS      C .                   DATA FIELD ATTRIBUTE
BLUEI   DS      0CL9 .              INPUT DATA FIELD
BLUEO   DS      CL9 .               OUTPUT DATA FIELD

REDL    DS      H   .                 INPUT DATA FIELD LENGTH
REDF    DS      0C .                  DATA FIELD FLAG
REDA    DS      C .                   DATA FIELD ATTRIBUTE
REDI    DS      0CL8 .              INPUT DATA FIELD
REDO    DS      CL8 .               OUTPUT DATA FIELD

AMBERL  DS      H   .                 INPUT DATA FIELD LENGTH
AMBERF  DS      0C .                  DATA FIELD FLAG
AMBERA  DS      C .                   DATA FIELD ATTRIBUTE
AMBERI  DS      0CL10 .             INPUT DATA FIELD
AMBERO  DS      CL10 .              OUTPUT DATA FIELD

ERRORL  DS      H   .                 INPUT DATA FIELD LENGTH
ERRORF  DS      0C .                  DATA FIELD FLAG
ERRORA  DS      C .                   DATA FIELD ATTRIBUTE
ERRORI  DS      0CL19 .             INPUT DATA FIELD
ERRORO  DS      CL19 .              OUTPUT DATA FIELD

MAPE      EQU   *       .             END OF MAP DEFINITION
          ORG
SAMPLET     EQU   *       .         END OF MAP SET DEFINITION
* * * END OF MAP DEFINITION * * *
```

Figure 10-9. Symbolic Storage Definition using LANG=ASM,MODE=INOUT

```
MAPI     DS     0C .                MAP ORIGIN

NAMEL    DS     H   .                 INPUT DATA FIELD LENGTH
NAMEF    DS     0C .                  DATA FIELD FLAG
         DS     C .                   DATA FIELD ATTRIBUTE
NAMEI    DS     CL18 .              DATA FIELD

*  START NEW DATA GROUP DATE
DATEL    DS     H   .                 INPUT GROUP FIELD LENGTH
DATEF    DS     0C .                  GROUP FIELD FLAG
         DS     C .                   GROUP FIELD ATTRIBUTE
DATEI    DS     0C .                  GROUP FIELD ORIGIN

MONTHI   DS     CL2 .               DATA FIELD

DAYI     DS     CL2 .               DATA FIELD

YEARI    DS     CL2 .               DATA FIELD

BLUEL    DS     H   .                 INPUT DATA FIELD LENGTH
BLUEF    DS     0C .                  DATA FIELD FLAG
         DS     C .                   DATA FIELD ATTRIBUTE
BLUEI    DS     CL1 .               DATA FIELD

REDL     DS     H   .                 INPUT DATA FIELD LENGTH
REDF     DS     0C .                  DATA FIELD FLAG
         DS     C .                   DATA FIELD ATTRIBUTE
REDI     DS     CL1 .               DATA FIELD

AMBERL   DS     H   .                 INPUT DATA FIELD
AMBERF   DS     0C .                  DATA FIELD FLAG
         DS     C .                   DATA FIELD ATTRIBUTE
AMBERI   DS     CL1 .               DATA FIELD

ERRORL   DS     H   .                 INPUT DATA FIELD LENGTH
ERRORF   DS     0C .                  DATA FIELD FLAG
         DS     C .                   DATA FIELD ATTRIBUTE
ERRORI   DS     CL19 .              DATA FIELD

         ORG
SAMPLET  EQU    *          .          END OF MAP SET DEFINITION
* * * END OF MAP DEFINITION * * *
```

Figure 10-10. Symbolic Storage Definition using LANG=ASM,MODE=IN

```
MAPO     DS     0C .                MAP ORIGIN

         DS     H   .                 INPUT DATA FIELD LENGTH
NAMEA    DS     C .                 | DATA FIELD ATTRIBUTE
NAMEO    DS     CL18 .              DATA FIELD

*  START NEW DATA GROUP DATE
         DS     H   .               | INPUT GROUP FIELD LENGTH
DATEA    DS     CL1 .               | GROUP FIELD ATTRIBUTE
DATEO    DS     0C .                | GROUP FIELD ORIGIN

MONTHO   DS     CL2 .               DATA FIELD

DAYO     DS     CL2 .               DATA FIELD

YEARO    DS     CL2 .               DATA FIELD

         DS     H   .               | INPUT DATA FIELD LENGTH
BLUEA    DS     C .                 | DATA FIELD ATTRIBUTE
BLUEO    DS     CL9 .               DATA FIELD

         DS     H   .               | INPUT DATA FIELD LENGTH
REDA     DS     C .                 | DATA FIELD ATTRIBUTE
REDO     DS     CL8 .               DATA FIELD

         DS     H   .               | INPUT DATA FIELD LENGTH
AMBERA   DS     C .                 | DATA FIELD ATTRIBUTE
AMBERO   DS     CL10 .              DATA FIELD

         DS     H   .               | INPUT DATA FIELD LENGTH
ERRORA   DS     C .                 | DATA FIELD ATTRIBUTE
ERRORO   DS     CL19 .              DATA FIELD

MAPE     EQU  *        .            END OF MAP DEFINITION
         ORG
SAMPLET    EQU  *          .         | END OF MAP SET DEFINITION
* * * END OF MAP DEFINITION * * *
```

Figure 10-11. Symbolic Storage Definition using LANG=ASM,MODE=OUT

```
01  MAPI.
    02  NAMEL    COMP    PIC  S9(4).
    02  NAMEA    PICTURE X.
    02  FILLER REDEFINES NAMEA.
      03  NAMEF     PICTURE X.
    02  NAMEI    PIC X(18).
    02  FILLER PIC X.
    02  DATEL    COMP    PIC  S9(4).
    02  DATEA    PICTURE X.
    02  FILLER REDEFINES DATEA.
      03  DATEF     PICTURE X.
    02  DATEI.
      03  MONTHI    PIC X(2).
      03  DAYI      PIC X(2).
      03  YEARI     PIC X(2).
      03  FILLER    PIC X.
    02  BLUEL    COMP    PIC  S9(4).
    02  BLUEA    PICTURE X.
    02  FILLER REDEFINES BLUEA.
      03  BLUEF     PICTURE X.
    02  BLUEI    PIC X(9).
    02  REDL     COMP    PIC  S9(4).
    02  REDA     PICTURE X.
    02  FILLER REDEFINES REDA.
      03  REDF      PICTURE X.
    02  REDI    PIC X(8).
    02  FILLER PIC X.
    02  AMBERL  COMP    PIC  S9(4).
    02  AMBERA  PICTURE X.
    02  FILLER REDEFINES AMBERA.
      03  AMBERF    PICTURE X.
    02  AMBERI  PIC  X(10).
    02  FILLER PIC X.
    02  ERRORL  COMP    PIC  S9(4).
    02  ERRORA  PICTURE X.
    02  FILLER REDEFINES ERRORA.
      03  ERRORF    PICTURE X.
    02  ERRORI  PIC  X(19).
01  MAPO REDEFINES MAPI.
    02  FILLER PICTURE X(3).
    02  NAMEO  PIC X(18).
    02  FILLER PIC X.
    02  FILLER PICTURE X(3).
    02  DATEO.
      03  MONTHO  PIC X(2).
      03  DAYO  PIC X(2).
      03  YEARO   PIC X(2).
      03  FILLER PIC X.
    02  FILLER PICTURE X(3).
    02  BLUEO  PIC X(9).
    02  FILLER PICTURE X(3).
    02  REDO  PIC X(8).
    02  FILLER PIC X.
    02  FILLER PICTURE X(3).
    02  AMBERO  PIC X(10).
    02  FILLER PIC X.
    02  FILLER PICTURE X(3).
    02  ERRORO  PIC X(19).
```

Figure 10-12. Symbolic Storage Definition using LANG=COBOL,MODE=INOUT

```
01  MAPI.
    02  NAMEL    COMP    PIC  S9(4).
    02  NAMEF    PICTURE X.
    02  NAMEI   PIC X(18).
    02  FILLER PIC X.
    02  DATEL    COMP    PIC  S9(4).
    02  DATEF    PICTURE X.
    02  DATEI.
       03  MONTHI  PIC X(2).
       03  DAYI  PIC X(2).
       03  YEARI  PIC X(2).
       03  FILLER PIC X.
    02  BLUEL    COMP    PIC  S9(4).
    02  BLUEF    PICTURE X.
    02  BLUEI  PIC X(1).
    02  REDL     COMP    PIC  S9(4).
    02  REDF     PICTURE X.
    02  REDI   PIC X(1).
    02  AMBERL  COMP    PIC  S9(4).
    02  AMBERF  PICTURE X.
    02  AMBERI  PIC X(1).
    02  ERRORL  COMP    PIC  S9(4).
    02  ERRORF  PICTURE X.
    02  ERRORI  PIC X(19).
```

Figure 10-13. Symbolic Storage Definition using LANG=COBOL,MODE=IN

```
01  MAPO.
    02  FILLER PICTURE X(2).
    02  NAMEA    PICTURE X.
    02  NAMEO  PIC X(18).
    02  FILLER PIC X.
    02  FILLER PICTURE X(2).
    02  DATEA    PICTURE X.
    02  DATEO.
       03  MONTHO  PIC X(2).
       03  DAYO  PIC X(2).
       03  YEARO  PIC X(2).
       03  FILLER PIC X.
    02  FILLER PICTURE X(2).
    02  BLUEA    PICTURE X.
    02  BLUEO  PIC X(9).
    02  FILLER PICTURE X(2).
    02  REDA     PICTURE X.
    02  REDO  PIC X(8).
    02  FILLER  PIC X.
    02  FILLER PICTURE X(2).
    02  AMBERA      PICTURE X.
    02  AMBERO  PIC X(10).
    02  FILLER  PIC X.
    02  FILLER PICTURE X(2).
    02  ERRORA      PICTURE X.
    02  ERRORO  PIC X(19).
```

Figure 10-14. Symbolic Storage Definition using LANG=COBOL,MODE=OUT

```
DECLARE 1 MAPI BASED (BMSMAPER),
     2  NAMEL    FIXED BINARY (15,0),
     2  NAMEA    CHARACTER (1),
     2  NAMEI  CHARACTER (18),
     2  DATEL   FIXED BINARY (15,0),
     2  DATEA   CHARACTER (1),
     2  DATEI,
       3  MONTHI  CHARACTER (2),
       3  DAYI  CHARACTER (2),
       3  YEARI  CHARACTER (2),
     2  BLUEL   FIXED BINARY (15,0),
     2  BLUEA   CHARACTER (1),
     2  BLUEI  CHARACTER (9),
     2  REDL    FIXED BINARY (15,0),
     2  REDA    CHARACTER (1),
     2  REDI  CHARACTER (8),
     2  AMBERL   FIXED BINARY (15,0),
     2  AMBERA   CHARACTER (1),
     2  AMBERI  CHARACTER (10),
     2  ERRORL   FIXED BINARY (15,0),
     2  ERRORA   CHARACTER (1),
     2  ERRORI  CHARACTER (19),
     2 FILL0271  CHARACTER (1);
DECLARE 1 MAPO BASED (BMSMAPER),
     2  DFHMS16 FIXED BINARY (15,0),
     2  NAMEF CHARACTER (1),
     2  NAMEO  CHARACTER (18),
     2  DFHMS17 FIXED BINARY (15,0),
     2  DATEF CHARACTER (1),
     2  DATEO,
       3  MONTHO  CHARACTER (2),
       3  DAYO  CHARACTER (2),
       3  YEARO  CHARACTER (2),
     2  DFHMS18 FIXED BINARY (15,0),
     2  BLUEF CHARACTER (1),
     2  BLUEO  CHARACTER (9),
     2  DFHMS19 FIXED BINARY (15,0),
     2  REDF CHARACTER (1),
     2  REDO  CHARACTER (8),
     2  DFHMS20 FIXED BINARY (15,0),
     2  AMBERF CHARACTER (1),
     2  AMBERO  CHARACTER (10),
     2  DFHMS21 FIXED BINARY (15,0),
     2  ERRORF CHARACTER (1),
     2  ERRORO  CHARACTER (19),
     2 FILL0271  CHARACTER (1);
/* END OF MAP DEFINITION */
```

Figure 10-15. Symbolic Storage Definition using LANG=PLI,MODE=INOUT

```
DECLARE 1 MAPI BASED (BMSMAPBR),
     2  NAMEL    FIXED BINARY (15,0),
     2  NAMEF    CHARACTER (1),
     2  NAMEI  CHARACTER (18),
     2  DATEL    FIXED BINARY (15,0),
     2  DATEF    CHARACTER (1),
     2  DATEI,
       3  MONTHI  CHARACTER (2),
       3  DAYI  CHARACTER (2),
       3  YEARI  CHARACTER (2),
     2  BLUEL    FIXED BINARY (15,0),
     2  BLUEF    CHARACTER (1),
     2  BLUEI  CHARACTER (1),
     2  REDL    FIXED BINARY (15,0),
     2  REDF    CHARACTER (1),
     2  REDI  CHARACTER (1),
     2  AMBERL   FIXED BINARY (15,0),
     2  AMBERF   CHARACTER (1),
     2  AMBERI  CHARACTER (1),
     2  ERRORL   FIXED BINARY (15,0),
     2  ERRORF   CHARACTER (1),
     2  ERRORI  CHARACTER (19),
     2 FILL0292  CHARACTER (1);
/* END OF MAP DEFINITION */
```

Figure 10-16. Symbolic Storage Definition using LANG=PLI,MODE=IN

```
DECLARE 1 MAPO BASED (BMSMAPER),
     2  DFHMS22 FIXED BINARY (15,0),
     2  NAMEA    CHARACTER (1),
     2  NAMEO  CHARACTER (18),
     2  DFHMS23 FIXED BINARY (15,0),
     2  DATEA    CHARACTER (1),
     2  DATEO,
       3  MONTHO  CHARACTER (2),
       3  DAYO  CHARACTER (2),
       3  YEARO  CHARACTER (2),
     2  DFHMS24 FIXED BINARY (15,0),
     2  BLUEA    CHARACTER (1),
     2  BLUEO  CHARACTER (9),
     2  DFHMS25 FIXED BINARY (15,0),
     2  REDA    CHARACTER (1),
     2  REDO  CHARACTER (8),
     2  DFHMS26 FIXED BINARY (15,0),
     2  AMBERA   CHARACTER (1),
     2  AMBERO  CHARACTER (10),
     2  DFHMS27 FIXED BINARY (15,0),
     2  ERRORA    CHARACTER (1),
     2  ERRORO  CHARACTER (19),
     2 FILL0320  CHARACTER (1);
/* END OF MAP DEFINITION */
```

Figure 10-17. Symbolic Storage Definition using LANG=PLI,MODE=OUT

CHAPTER 11. APPLICATION PROGRAMMING CONSIDERATIONS

TERMINAL-ORIENTED TASK IDENTIFICATION

When CICS/VS receives input from a terminal to which no task is attached, the system has to determine which task should be initiated. The methods by which the user can specify the task to be attached and the sequence in which the system checks these specifications is as follows (see also Figure 11-1).

Test 1:
- a) Is this terminal of a type supported by the basic mapping support terminal paging facility?
- b) Is the input a paging command? (The terminal operator enters paging commands as specified by the PGRET, PGCHAIN, PGPURGE, and PGCOPY operands of the DFHSIT macro instruction by the system programmer. See the CICS/VS System Programmer's Reference Manual.)

If yes, task attached: the task identified by transaction identification CSPG, to process paging commands.

Test 2: Is a task specified by the TRANSID operand of the DFHTCT macro instruction creating a terminal control table entry for the terminal?

If yes, task attached: the task specified by the TRANSID operand used in the creation of the terminal control table entry.

Test 3: Is a task specified by the TRANSID operand of a DFHPC TYPE=RETURN macro instruction (or by the application program moving the task name into TCANXTID)?

If yes, task attached: the task specified by the TRANSID operand of the last DFHPC TYPE=RETURN macro instruction containing a TRANSID operand that was issued by the task.

Test 4:
- a) Is the terminal a 3270?
- b) Is the input a PA key, PF key, or LPA (light pen attention)?
- c) Is this input (PAn, PFn, or LPA) specified by the TASKREQ operand of a DFHPCT TYPE=ENTRY macro instruction? (See the CICS/VS System Programmer's Reference Manual.)

If yes, task attached: the task specified by the TASKREQ operand of same DFHPCT TYPE=ENTRY macro instruction.

Test 5: Is a valid transaction identification specified by the first one to four characters of the terminal input?

If yes, task attached: the task whose transaction identification is specified in the terminal input.

If none of the above tests is met, an invalid transaction identification exists. Message DFH2001 is sent to the terminal.

Note: The 3735 Programmable Buffered Terminal takes exception to this sequence when operating in inquiry mode. The test of input from the terminal (item 5 above) is given first priority.

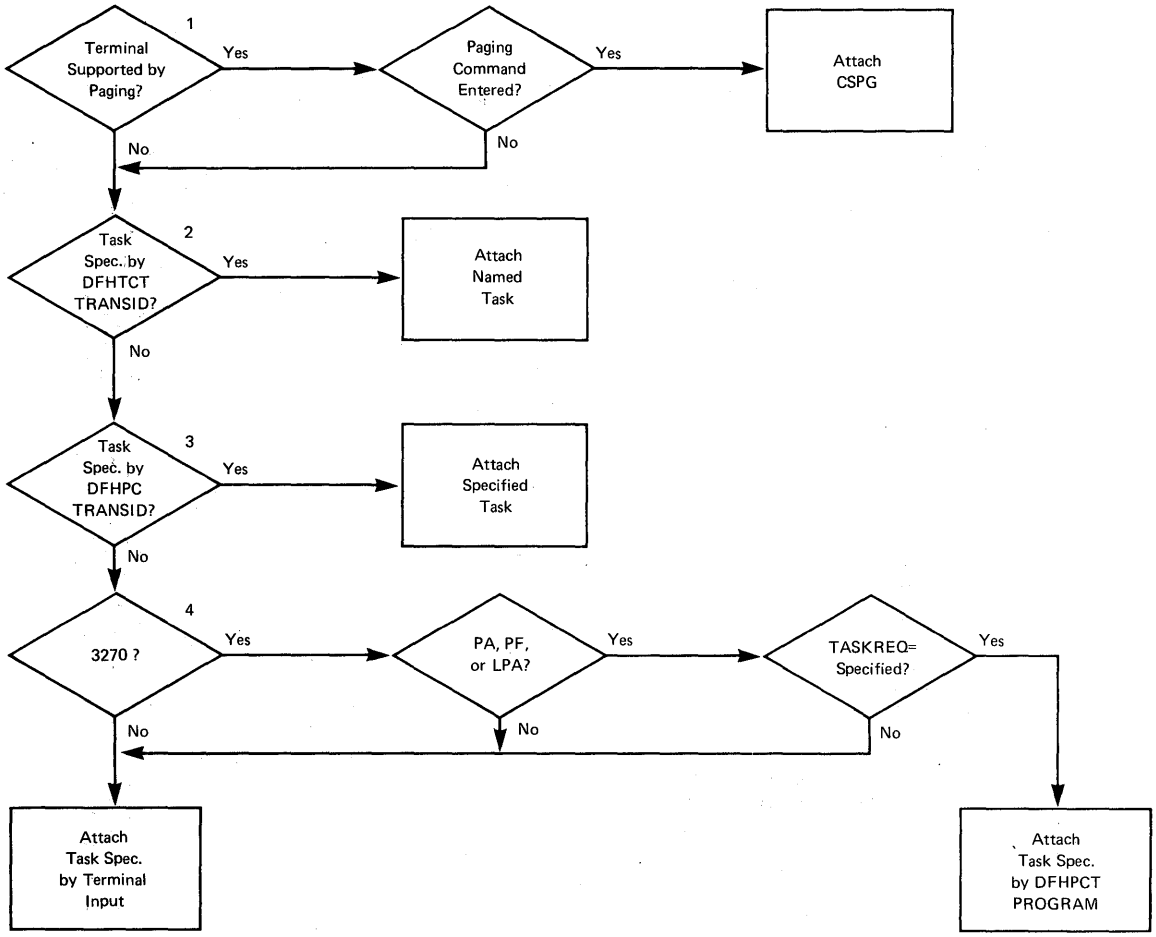


Figure 11-1. CICS/VS Terminal-Oriented Task Identification

PROGRAMMABLE DEVICE CONSIDERATIONS

When BTAM is used by CICS/VS for programmable binary synchronous communication line management, CICS/VS initializes the communication line with a BTAM read initial (TI); the terminal response must be a write initial (TI) or the equivalent. If a user-written application program then issues a read, CICS/VS issues a read continue (TT) to that line; if the application program then issues a write, CICS/VS issues a read interrupt (RVI) to that line. If end of transmission (EOT) is not received on the RVI, CICS/VS issues a read continue (TT) until the EOT is received.

The programmable terminal response to a read interrupt must be "end of transmission" (EOT), except that the EOT response may be preceded by writes to exhaust the contents of output buffers so long as the input buffer size (specified by the system programmer during preparation of the terminal control table) is not exceeded by this data. CICS/VS issues a read continue until it receives an EOT, or until the input message exceeds the size of the input buffer (which is an error condition).

After receiving an EOT, CICS/VS issues a write initial (TI) or the equivalent (depending on the type of line). The programmable terminal response must be a read initial (TI) or the equivalent.

If another write is issued by the application program, CICS/VS issues a write continue (TT) to that line. If the application program issues a read after it has issued a write, CICS/VS turns the line around with a write reset (TR). (CICS/VS does not recognize a read interrupt.)

To ensure that binary synchronous terminals (for example, System/370, 1130, 2780) remain coordinated, CICS/VS processes the data collection or data transmission transaction on a line to completion before it polls other terminals on that line.

The programmable terminal actions required for the activity sequence just discussed, with the corresponding user application program macro instructions and CICS/VS actions, are summarized below.

Application Program	CICS/VS (note 1)	Programmable Terminal Program
	Read initial (TI)	Write initial (TI)
DFHTC TYPE=READ	Read continue (TT)	Write continue (TT)
DFHTC TYPE=WRITE (note 2)	Read interrupt (RVI)	Write reset (TR), or
(note 3)	Read continue (TT)	Write continue Write reset
	Write initial (TI)	Read initial (TI)
DFHTC TYPE=WRITE	Write continue (TT)	Read continue (TT)
DFHTC TYPE=READ (note 4)	Write reset (TR) Read initial (TI)	Read continue (TT) Write initial (TI)

Notes:

1. CICS/VS issues the macro instruction shown, or, if the line is switched, the equivalent. The user-written programmable terminal program must issue the equivalent of the BTAM operation shown.

2. An RVI sequence is indicated by the DECFLAGS field of the data extent control block (DECB) being set to X'02' and a completion code of X'7F' being returned to the event control block (ECB).

3. The read continue is issued only if the EOT character is not received on the read interrupt.

4. Write reset is issued only for point-to-point terminals.

Input data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the TIOA but are not included in the data length (TIOATDL). The CICS/VS application programmer should be aware that characters such as NL, CR, LF, and EM are passed in the TIOA as data.

3735 CONSIDERATIONS

The 3735 Programmable Buffered Terminal may be serviced by CICS/VS in response to terminal-initiated input, or as a result of an automatic or time-initiated transaction. Both are explained below.

3735 Transactions - Autoanswer

The 3735 transaction is attached by CICS/VS upon receipt of input from a 3735. Data is passed to the application program in 476-byte blocks; each block (one buffer) may contain multiple logical records. The final block may be shorter than 476 bytes; however, zero-length final blocks are not passed to the application program. If the block contains multiple logical records, the application program must perform any necessary deblocking functions and the gathering of partial logical records from consecutive reads.

It is recommended that the user spool input data from a 3735 to an intermediate data set (for example, an intrapartition destination) to ensure that all data has been captured before deblocking and processing that data.

The application program must follow 3735 conventions and read to end-of-file before attempting to write FDPs (form description programs) or data to the 3735. For this reason, the EOF=symbolic address operand must be used with each DFHTC TYPE=READ request. When the EOF branch is taken, the user may begin to write FDPs or data to the 3735, or, optionally, request CICS/VS to disconnect the line.

It is possible that the 3735 will transmit the end-of-file condition immediately upon connection of the line. For this reason the user must code the initialization request (DFHTC EOF=symbolic address) before issuing any other terminal control requests.

The user is responsible for formatting all special message headers for output to the 3735 (for example, SELECTRIC, POWERDOWN). If FDPs are to be transmitted to a 3735 with ASCII transmission code, the NOTRANSLATE operand must be included in the DFHTC TYPE=WRITE request for each block of FDP records.

The user must issue a DFHTC TYPE=DISCONNECT macro instruction when all output has been transmitted to the 3735. If the application program ends during batch write mode prior to issuing the DISCONNECT request, CICS/VS forces a 3735 "receive abort" condition and all data just transmitted is ignored by the 3735.

3735 Transactions - Autocall

In automatic and time-initiated transactions, all considerations stated above except use of the DFHTC EOF=symbolic address macro instruction apply when CICS/VS dials a 3735. The DFHTC EOF=symbolic address macro instruction is not used.

CICS/VS connects the line and allows the user to indicate the direction of data transfer by means of the first terminal control request. If this first request is a WRITE and the 3735 has data to send, the 3735 causes the line to be disconnected.

3740 CONSIDERATIONS

The 3740 Data Entry System may be serviced by CICS/VS as a batch or inquiry mode application. Considerations for both modes are described in the following paragraphs.

Batch Mode Applications

In batch mode, the 3740 sends multiple files of data to CICS/VS during a single transmission. All input data files must be sent to

CICS/VS before the 3740 is able to receive data from CICS/VS. When able to receive, the 3740 accepts multiple files of data in a single transmission. To communicate in this manner, a means is provided in the DFHTC macro instruction for identifying end-of-file, end-of-input, and end-of-output conditions.

The ENDFILE=symbolic address operand in the DFHTC TYPE=READ macro instruction provides the means for the CICS/VS application program to determine the boundaries for each file received from the 3740. When the end-of-file condition occurs (defined by receipt of ETX), control is passed to the address specified. The TIOA contains no valid data at this time.

The ENDINPT=symbolic address operand in the DFHTC TYPE=READ macro instruction provides the means for the application program to determine when the 3740 has completed transmission of all data. When end-of-input (EOT) is received, control is passed to the address specified. The TIOA contains no valid data at this time. Once end-of-input is received, the application program must not issue any additional READs. An end-of-input condition may exist upon entry to the application program. Therefore, a DFHTC TYPE=READ with ENDINPT=symbolic address should always be issued before attempting any terminal operations.

When sending data to the 3740, the DFHTC TYPE=ENDFILE macro instruction must be issued after each file to signal the end-of-file (ETX) condition to the 3740. The DFHTC TYPE=ENDOUTPUT macro instruction should be issued after all data has been sent to the 3740 (EOT) and must be immediately preceded by a DFHTC TYPE=ENDFILE macro instruction. Once end-of-output is signalled in this manner, no additional WRITEs should be issued. The WRITE, ENDFILE, and ENDOUTPUT parameters may be combined in the DFHTC macro instruction. For example, a DFHTC TYPE=(WRITE, ENDFILE) causes a write operation followed by an end-of-file signal. A DFHTC TYPE=(WRITE, ENDFILE, ENDOUTPUT) causes a write operation, an end-of-file signal, and then an end-of-output signal. A DFHTC TYPE=(ENDFILE, ENDOUTPUT) causes an end-of-file signal followed by an end-of-output signal. The placement of the parameter within the macro instruction has no effect on the sequence.

The DFHPC TYPE=RETURN macro instruction with TRANSID=transaction code may be used in batch mode to return program control. It may not be used in batch write mode.

Inquiry Mode Applications

In inquiry mode, the application program must send a block of data to the 3740 for each block of data received; otherwise, the 3740 appears similar to the 2741 for inquiry applications. The DFHTC parameters discussed under "Batch Mode Applications" do not apply to inquiry mode operations.

SYSTEM/7 CONSIDERATIONS

The implementation of System/7 support treats the System/7 as any other programmable terminal. Transactions are normally initiated from the System/7 by issuing a four-character transaction code as in the following example:

```
          .
          .
          .
        @XMIT   TRNID                   TRANSMIT TRANSACTION CODE
        PBER    ERROR                   BRANCH IF CONDITION ERROR CODE
        PLEX                            WAIT FOR COMPLETION
          .
          .
          .
TRNID   #IOLT   3,CHECK,/0000,TRAN,2
*       #IOLT                           GENERATE I/O LIST
*               3,                      RETURN CONTROL ON INTERRUPT
*                                       LEVEL 3
*               CHECK,                  RETURN CONTROL AT LOCATION CHECK
*               /0000,                  TRANSMIT MESSAGE IN BCD MODE
*               TRAN,                   MESSAGE LOCATED AT TRAN
*               2                       MESSAGE TWO WORDS LONG
TRAN    PEQU    *                       TRANSACTION ID
        PDC     /A6D2                   ='TR'
        PDC     /CA0E                   ='N7'
          .
          .
CHECK   PEQU    *                       TEST FOR SUCCESSFUL COMPLETION
          .
          .
```

In this example, the transaction identification is transmitted in BCD mode. Pseudobinary mode can be used only while communicating with an active CICS/VS transaction; it cannot be used to initiate the transaction. Note that the message length is given as the number of words to be transmitted (not as the number of characters).

When a transaction is initiated on a System/7, CICS/VS services only that System/7 for the duration of the transaction; to ensure efficient use of the line, any other System/7s on the same line are locked out for the duration of the transaction. Therefore, CICS/VS application programs for the multipoint System/7 should be designed so that their execution is of short duration.

It is an MSP/7 standard that the first word (two characters) of every message received by the System/7 be an identification word. Note, however, that all identification words beginning with "@" (X'20') are reserved for future use by CICS/VS.

When the PSEUDOBIN parameter is specified as part of an input request (for example, DFHTC TYPE=(READ,PSEUDOBIN)), the length of the TIOA provided by the application program must be at least twice that of the data to be read. For example, if twenty System/7 words (40 bytes) are to be read, the data area of the TIOA must be at least 80 bytes in length.

When the PSEUDOBIN parameter is specified as part of an output request, terminal control always obtains a new TIOA and frees the old TIOA unless SAVE was specified. Therefore, on a DFHTC TYPE=(WRITE,READ,PSEUDOBIN) request, the application program must reload the TIOA address (from TCTTEDA) to access the input data from the System/7.

In the case of a System/7 on a dial-up (switched) line, the System/7 application program must initially transmit a four-character terminal identification. (This terminal identification is generated during preparation of the TCT through use of the DFHTCT TYPE=TERMINAL, TRMIDNT=parameter specification.) CICS/VS responds with either a "ready" message, indicating that the terminal identification is valid and that the System/7 may proceed as if it were on a leased line, or an INVALID TERMINAL IDENTIFICATION message, indicating that the terminal

identification sent by the System/7 did not match the TRMIDNT=parameter specification.

Whenever CICS/VS initiates the connection to a dial-up System/7, CICS/VS writes a null message consisting of three idle characters prior to starting the transaction. BTAM error routines cause a data check message to be recorded on the CICS/VS (host) system console if there is no program resident in the System/7 capable of supporting the Asynchronous Communication Control Adapter (ACCA). This is normal when the task to be initiated by CICS/VS is to IPL the System/7. Although the data check message is printed, CICS/VS ignores the error and continues normal processing. If a program capable of supporting the ACCA is resident in the System/7 at the time of this message transmission, no data check occurs.

When a disconnect is issued to a dial-up System/7, the 'busy' bit is sometimes left on in the interrupt status word of the ACCA. If the line connection is reestablished by dialing from the System/7 end, the 'busy' condition of the ACCA prevents message transmission from the System/7. To overcome this problem, the System/7 program must reset the ACCA after each disconnect and before message transmission is attempted. This can be accomplished by issuing the following instruction:

```
PWRI    0,8,3,0     RESET ACCA
```

This procedure is not necessary when the line is reconnected by CICS/VS (that is, by an automatically initiated transaction).

NONPROGRAMMABLE-DEVICE CONSIDERATIONS

This section presents factors to be considered by the application programmer who designs applications for nonprogrammable terminals.

2260/2265 PROGRAMMING CONSIDERATIONS

The following is an example of the coding required to write data to a 2260/2265 terminal and specify the screen line address at which the write is to begin:

```
DFHTC TYPE=WRITE,           WRITE DATA TO A TERMINAL SCREEN    *
      LINEADR=10            STARTING AT THIS SCREEN LINE
```

The LINEADR operand is used to specify that writing is to begin on a specific line of a 2260 or 2265 screen. It is the responsibility of the application programmer to provide the hexadecimal equivalent of a line number in the range 1-12 (F0-FB) for the 2260 or 1-15 (F0-FE) for the 2265. He can accomplish this in either of two ways: (1) by including the LINEADR=number operand in the DFHTC macro instruction, or (2) by coding a single instruction, prior to issuing the DFHTC macro instruction, that places the line number in the TIOALAC field of the current TIOA. If the latter method is used, the LINEADR=YES operand must be included in the DFHTC macro instruction.

The following are examples of the coding required to write data to a 2260/2265 terminal and dynamically determine the screen line address at which the write is to begin.

For Assembler language:

```
        MVI   TIOALAC,X'F0'       WRITE STARTING AT SCREEN LINE 1
              .
              .
              .
        DFHTC TYPE=WRITE,         WRITE DATA TO A TERMINAL SCREEN    *
              LINEADR=YES         STARTING LINE ALREADY SPECIFIED
```

For ANS COBOL:

```
        MOVE 240 TO TIOALAC.      NOTE PLACE STARTING LINE IN TIOA.
              .
              .
              .
        DFHTC TYPE=WRITE,         WRITE DATA TO A TERMINAL SCREEN    *
              LINEADR=YES         STARTING LINE ALREADY SPECIFIED
```

For PL/I:

```
        TIOALAC=240;              /*START WRITE AT SCREEN LINE 1*/
              .
              .
              .
        DFHTC TYPE=WRITE,         WRITE DATA TO A TERMINAL SCREEN    *
              LINEADR=YES         STARTING LINE ALREADY SPECIFIED
```

3270 OPERATING IN 2260 COMPATIBILITY MODE

The following example shows how to write data to a 3270 terminal operating in 2260 compatibility mode, and how to specify the screen line address at which the write is to begin:

```
        DFHTC TYPE=WRITE,         WRITE DATA TO A TERMINAL SCREEN    *
              LINEADR=10          STARTING AT THIS SCREEN LINE
```

The following examples show how to write data to a 3270 terminal operating in 2260 compatibility mode, beginning at a screen line address placed in TIOALAC prior to issuing the write request.

For Assembler language:

```
        MVI   TIOALAC,X'F0'       WRITE STARTING AT SCREEN LINE 1
              .
              .
              .
        DFHTC TYPE=WRITE,         WRITE DATA TO A TERMINAL SCREEN    *
              LINEADR=YES         STARTING LINE ALREADY SPECIFIED
```

For ANS COBOL:

```
        MOVE 240 TO TIOALAC.      NOTE PLACE STARTING LINE IN TIOA.
              .
              .
              .
        DFHTC TYPE=WRITE,         WRITE DATA TO A TERMINAL SCREEN    *
              LINEADR=YES         STARTING LINE ALREADY SPECIFIED
```

For PL/I:

```
TIOALAC=240;                /*START WRITE AT SCREEN LINE 1*/
      .
      .
      .
DFHTC TYPE=WRITE,           WRITE DATA TO A TERMINAL SCREEN    *
      LINEADR=YES           STARTING LINE ALREADY SPECIFIED
```

2770/2780 PROGRAMMING CONSIDERATIONS

The 2770 Data Communication System and 2780 Data Transmission Terminal recognize a read interrupt and respond by transmitting the contents of the I/O buffer. After the contents of the buffer have been transmitted, the 2770 or 2780 responds to the next read continue with an EOT. If the I/O buffer is empty, the 2770 or 2780 transmits an EOT. CICS/VS issues a read interrupt and read continue to relinquish use of the line and to enable the application program to write to the 2770 or 2780.

Input from a 2770 or 2780 consists of one or more logical records. CICS/VS provides one logical record for each read request to the application program. Note that the size of a logical record cannot exceed the size of the I/O buffer. If the input spans multiple buffers, multiple reads must be issued by the application program.

Output to a 2780 requires that the application programmer insert the appropriate "escape sequence" for component selection associated with the output message. (For programming details, see the publication Component Description: IBM 2780 Data Transmission Terminal.)

The 2265 component of the 2770 Data Communication System is controlled by data stream characters, not BTAM macro instructions. Therefore, the user should provide the appropriate screen control characters in the TIOA. The DFHTC TYPE=ERASE parameter and LINEADR operand do not apply.

For 2770 and 2780 input, data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the TIOA but are not included in the data length (TIOATDL). The application programmer should be aware that such characters as NL, CR, and LF are passed in the TIOA as data.

2980 PROGRAMMING CONSIDERATIONS

CICS/VS provisions for support of the 2980 General Banking Terminal System are described in the following paragraphs. Fields that contain information of interest to the application program are pointed out.

Passbook Control

Two one-byte fields of the terminal control table terminal entry (TCTTE) may be interrogated by an application program servicing passbook requests from the 2980. These fields are:

- TCTTETAB, which contains the binary representation of the number of tabs necessary to position the print element to the correct passbook area.
- TCTTEPCF, which contains the indicators (flags) necessary for passbook control operations. The indicators TCTTEPCR and TCTTEPCW indicate whether or not the passbook is present on a read or a

write operation, respectively. The same indicators are used to indicate the presence of the Auditor key on the 2980 Model 2.

By testing indicators TCTTEPCR and TCTTEPCW, the application program can maintain positive control with regard to the absence or presence of a passbook during an update operation. However, care must be taken not to alter these indicators because unpredictable results may occur.

If the passbook is present on a read (entry) operation, the TCTTEPCR indicator is turned on (set to a binary one) by CICS/VS. In this case, the application program generally issues a write operation back to the passbook area to update the passbook. After the write operation, the application program must check the TCTTEPCW indicator to ensure that the passbook was present at the time the write occurred. If the TCTTEPCW indicator is off (set to a binary zero), the passbook was not present and the write operation did not occur. However, the data sent to the terminal (and not printed because of the "no passbook" condition) is returned to the application program in its original form for subsequent retransmission.

When the "no passbook" condition occurs on a write, CICS/VS allows an immediate write to the terminal. The application program should generally write an error message to the journal area of the terminal informing the 2980 operator of this error condition. To allow the operator to insert the required passbook, CICS/VS automatically causes the transaction to wait 23.5 seconds before continuing execution.

After regaining control from CICS/VS following the writing of the error message, the application program can attempt another write to the passbook area after ensuring that the print element is positioned correctly in the passbook area. This is generally accomplished by issuing two carrier returns followed by the number of tabs required to move the print element to the correct position. (The correct number of tabs can be acquired from TCTTETAB.)

If the TCTTEPCW indicator is off following the second attempt to write to the passbook area, the application program can send another error message or take some alternative action (for example, place the terminal "out of service").

In summary, all writes to the passbook area are conditional. That is, a passbook must be present before a write can be executed successfully. Therefore, a read operation cannot be combined with a passbook write. For example, a DFHTC TYPE=(WRITE,READ,WAIT) macro instruction is an invalid request for 2980 terminal services involving the passbook area. A DFHTC TYPE=PASSBK macro instruction is permissible because it implies only WRITE,WAIT.

Note: The application programmer should not insert shift characters in output data, since this is done automatically by CICS/VS. CICS/VS removes shift characters from input data.

Segmented Writes Control

Segmented writes are supported for both the journal area and the passbook area. Journal area segmented writes are limited in length by the hexadecimal halfword value that the user stores in TIOATDL. Passbook segmented writes are limited to a one-line logical write to ensure positive control when spacing (indexing) past the bottom of the passbook.

For example, consider a 2972 buffer length of 48 and a 2980 Model 4 logical write (print) area of 100 characters per line. The application program can write a logical record (DFHTC TYPE=PASSBK) of

100 characters to this area; CICS/VS automatically segments the record to adjust to the buffer size. The application program must insert the passbook indexing character (X'25') as the last character written in one logical write to the passbook area. This is done to control passbook indexing and thereby achieve positive control of passbook presence.

If the message contains embedded passbook index characters and segmentation is necessary because of the logical length of the message, the write terminates if the passbook spaces past the bottom of the passbook; the remaining segments are not printed.

Data Handling

SHIFT CHARACTERS: Shift characters are handled by the terminal control program and are of no concern to the application programmer. They are stripped from input messages and added to output messages as required. Data can be written in any mix of uppercase, lowercase, or special characters. (See the 2980 Translate Tables in Appendix E.)

JOURNAL INDEXING: Journal indexing is the responsibility of the application programmer. Carriage returns (X'15') may be inserted anywhere in the logical message.

PASSBOOK INDEXING: Passbook indexing necessitates special consideration by the application programmer to control bottom-line printing on the passbook. (See "Passbook Control" and "Segmented Writes Control.")

TAB CHARACTERS: The tab character (X'05') is also controlled by the application programmer. As stated above, the number of tabs required to position the print element to the first position of the passbook is available at TCTTETAB. This value is specified by the system programmer when generating the terminal control table and may be unique to each terminal. Other tab characters are inserted as needed to control output format.

MISCELLANEOUS CHARACTERS: Turn page, message light, openchute, and special banking characters can be used by the application programmer as needed. (See the 2980 Translate Tables in Appendix E.)

AUDITOR KEY MODEL 2: Presence of the Auditor key is controlled through use of the DFHTC TYPE=PASSBK macro instruction and may be used in a manner similar to that for passbook control. (See "Passbook Control.")

2980 MODEL NUMBER: TCTTETM contains the 2980 model number expressed as a hexadecimal value (X'01', X'02', X'04'). Since CICS/VS uses the model number to select the correct translate table for each of the 2980 models, the application program should not alter this field.

COMMON BUFFER: Common buffer writes (DFHTC TYPE=CBUFF) are translated to the receiving TCTTE model character set. If more than one 2980 model type is connected to the 2972 Control Unit, the lengths are automatically truncated if they exceed the buffer size.

Writing High-Level-Language Programs

The high-level-language application programmer must be concerned with the following fields of the DFHTCTTE structure when writing programs to run on a 2980 General Banking Terminal System:

Field	Meaning
TCTTETAB	Number of tab characters (binary)
TCTTEPCF	Passbook control field
TCTTESID	Station identification
TCTTETID	Model 4 teller identification

As discussed under "Passbook Control," the application program is expected to examine TCTTETAB to determine the correct number of tab characters to place in output data. The following examples show how this can be done in ANS COBOL and PL/I programs.

For ANS COBOL:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DFH2980 COPY DFH2980.
   .
   .
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   02  TCTTEAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   02  TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   .
   .
01 DFHTCTTE COPY DFHTCTTE.
01 DFHTIOA COPY DFHTIOA.
   02  DATA PICTURE X(20).
   02  FILLER REDEFINES DATA.
     03  TAB1-1 PICTURE X.
     03  DATA1 PICTURE X(19).
   02  FILLER REDEFINES DATA.
     03  TAB1-2 PICTURE X.
     03  TAB2-2 PICTURE X.
     03  DATA2 PICTURE X(18).
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
   IF TCTTETAB = TAB-ONE GO TO ONETBCH.
   IF TCTTETAB = TAB-TWO GO TO TWOTBCH.
   .
   .
   .
ONETBCH.
   MOVE TABCHAR TO TAB1-1.
   MOVE TOTAL TO DATA1.
   .
   .
TWOTBCH.
   MOVE TABCHAR TO TAB1-2, TAB2-2.
   MOVE TOTAL TO DATA2.
   .
   .
```

For PL/I:

```
%INCLUDE DFHTIOA;
       2 DATA CHAR (20);
DECLARE 1 USERTIOA_1 BASED (TIOABAR),
       2 TIOAFILL CHAR (12),
       2 TAB1_1 CHAR (1),
       2 DATA1 CHAR (19);
DECLARE 1 USERTIOA_2 BASED (TIOABAR),
       2 TIOAFILL CHAR (12),
       2 TAB1_2 CHAR (1),
       2 TAB2_2 CHAR (1),
       2 DATA2 CHAR (18);
       .
       .
       .
%INCLUDE DFH2980;
       .
       .
       .
       IF (TCTTETAB = TAB_ONE) THEN GO TO ONETBCH;
       IF (TCTTETAB = TAB_TWO) THEN GO TO TWOTECH;
       .
       .
       .
ONETBCH:   TAB1_1 = TABCHAR;
           DATA1 = AMOUNT;
           .
           .
           .
TWOTBCH:   TAB1_2 = TABCHAR;
           TAB2_2 = TABCHAR;
           DATA2 = AMOUNT;
           .
           .
           .
```

In the ANS COBOL example, the structure DFH2980 is copied in the Working Storage Section; in the PL/I example, DFH2980 is included following the %INCLUDE statements for the based structures. DFH2980 contains constants that may be used when writing application programs for the 2980.

The application program is also expected to test the TCTTEPCF field to determine whether a passbook was present on a read or write. TCTTEPCR and TCTTEPCW are located in DFH2980 to aid in this testing.

To test the TCTTEPCF field in ANS COBOL, statements such as the following might be used:

```
MOVE TCTTEPCF TO HOLDPCF.
IF HOLDPCFB = (HOLDPCFB / TCTTEPCW) * TCTTEPCW
THEN GO TO BOOK-FOR-PRESENT-WRITE.
```

Substituting TCTTEPCR for TCTTEPCW allows the ANS COBOL programmer to test for the presence of a passbook on a read. (HOLDPCF and HOLDPCFB are also part of DFH2980.)

To test the TCTTEPCF field in PL/I, statements such as the following might be used:

```
IF (TCTTEPCF | TCTTEPCW) THEN GO TO
BOOK_PRESENT_WRITE;
```

Substituting TCTTEPCR for TCTTEPCW allows the PL/I programmer to test for the presence of a passbook on a read.

To test the station identification and to determine whether the normal station or alternate station is being used, values of the forms shown below are predefined in DFH2980:

STATION-#-A OR STATION-#-N (for ANS COBOL)

STATION_#_A OR STATION_#_N (for PL/I)

where # is an integer (0 through 9) and A and N signify alternate and normal stations. The values are one-byte character values and can be compared to TCTTESID in an IF statement.

To test the teller identification on a 2980 Model 4, the TCTTETID field is defined as a one-byte character value. It can be tested in an IF statement.

Thirty special characters are defined in DFH2980. Twenty-three of these can be referred to by the name SPECCHAR-X or SPECCHAR_X (for ANS COBOL or PL/I) where X is an integer (0 through 23). The seven other characters are defined with names that imply their usage, for example, TABCHAR. For further information on these thirty characters, see Appendix E.

The names defined in DFH2980 for ANS COBOL follow:

STATION-0-N	STATION-6-A	TAB-SIX	MSGLITE	SPECCHAR-11
STATION-0-A	STATION-7-N	TAB-SEVEN	BCKSPACE	SPECCHAR-12
STATION-1-N	STATION-7-A	TAB-EIGHT	TRNPGE	SPECCHAR-13
STATION-1-A	STATION-8-N	TAB-NINE	SPECCHAR-1	SPECCHAR-14
STATION-2-N	STATION-8-A	HOLDPCFB	SPECCHAR-2	SPECCHAR-15
STATION-2-A	STATION-9-N	DFHFILL	SPECCHAR-3	SPECCHAR-16
STATION-3-N	STATION-9-A	HOLDPCF	SPECCHAR-4	SPECCHAR-17
STATION-3-A	TAB-ZERO	TCTTEPCR	SPECCHAR-5	SPECCHAR-18
STATION-4-N	TAB-ONE	TCTTEPCW	SPECCHAR-6	SPECCHAR-19
STATION-4-A	TAB-TWO	TABCHAR	SPECCHAR-7	SPECCHAR-20
STATION-5-N	TAB-THREE	OPENCH	SPECCHAR-8	SPECCHAR-21
STATION-5-A	TAB-FOUR	JRNLCR	SPECCHAR-9	SPECCHAR-22
STATION-6-N	TAB-FIVE	PSBKCR	SPECCHAR-10	SPECCHAR-23

The names defined in DFH2980 for PL/I follow:

STATION_0_N	STATION_7_A	TCCTEPCR	SPECCHAR_7	SPECCHAR_22
STATION_0_A	STATION_8_N	TCTTEPCW	SPECCHAR_8	SPECCHAR_23
STATION_1_N	STATION_8_A	TABCHAR	SPECCHAR_9	
STATION_1_A	STATION_9_N	OPENCH	SPECCHAR_10	
STATION_2_N	STATION_9_A	JRNLCR	SPECCHAR_11	
STATION_2_A	TAB_ZERO	PSBKCR	SPECCHAR_12	
STATION_3_N	TAB_ONE	MSGLITE	SPECCHAR_13	
STATION_3_A	TAB_TWO	BCKSPACE	SPECCHAR_14	
STATION_4_N	TAB_THREE	TRNPGE	SPECCHAR_15	
STATION_4_A	TAB_FOUR	SPECCHAR_1	SPECCHAR_16	
STATION_5_N	TAB_FIVE	SPECCHAR_2	SPECCHAR_17	
STATION_5_A	TAB_SIX	SPECCHAR_3	SPECCHAR_18	
STATION_6_N	TAB_SEVEN	SPECCHAR_4	SPECCHAR_19	
STATION_6_A	TAB_EIGHT	SPECCHAR_5	SPECCHAR_20	
STATION_7_N	TAB_NINE	SPECCHAR_6	SPECCHAR_21	

7770 PROGRAMMING CONSIDERATIONS

Even though CICS/VS does not distinguish between special codes (characters) entered at an audio terminal (for example, the 2721 Portable Audio Terminal), an application program is not precluded from performing special functions upon encountering these codes. For example, the following special hexadecimal codes may be entered from a 2721 Portable Audio Terminal:

Key	Code
CALL END	37**
CNCL	18
#	3B** or 7B
VERIFY	2D
RPT	3D
EXEC	26**
F1	B1
F2	B2
F3	B3
F4	B4
F5	B5
00	A0
000	3B** or B0
IDENT	11, 12, 13, or 14 plus two other characters

For further information concerning the 2721, see the publication 2721 Portable Audio Terminal Component Description.

The following special hexadecimal codes may be entered from a Touch-Tone[1] telephone:

Key	Code
*	A0
#	3B** or B0

The * and # characters of a Touch-Tone telephone correspond to the 00 and 000 characters, respectively, on a 2721 Portable Audio Terminal.

---

[1]Trademark of the American Telephone & Telegraph Co.

The codes denoted by a double asterisk above cause a hardware interrupt and are in the terminal input/output area (TIOA) immediately following the data; the codes are not included in the data length.

Note: The # and 000 characters cause an end-of-inquiry (EOI) hardware interrupt (X'3B') unless the EOI Disable feature (#3540) is installed on the 7770 Audio Response Unit Model 3. In this case, at the option of the user, either or both of the # and 000 characters do not cause a hardware interrupt, are presented in the TIOA along with the rest of data, and are included in the data length.

If, after receiving at least one character from a terminal, no other characters have been received by the 7770 for a period of five seconds, the 7770 automatically generates an EOI hardware interrupt that ends the read operation.

2741 READ ATTENTION AND WRITE BREAK SUPPORT

Read Attention support may be generated in any CICS/OS/VS or CICS/DOS/VS system to permit response to terminal operator pressing of the Attention key (rather than the Return key) on a 2741 Communications Terminal after typing a message, or without typing a message if no data is to be entered. Write Break support may be generated in any CICS/OS/VS system to permit response to terminal operator pressing of the Attention key on a 2741 while receiving a message. The following features must be installed on the 2741:

- For Read Attention: Transmit Interrupt (7900)
- For Write Break: Receive Interrupt (4708)

Read Attention

If the terminal operator presses the Attention key on the 2741 after typing a message, it is recognized as a Read Attention if:

- Read Attention support is generated into the system (CICS/OS/VS or CICS/DOS/VS)
- The message is read by a DFHTC TYPE=READ,RDATT=symbolic address macro instruction (which has an implied WAIT)

When this occurs, control is transferred to a CICS/VS read attention exit routine, if generated into the system. This routine is a skeleton program that can be tailored by the system programmer to perform actions such as the following:

- Perform some data analysis or modication on a Read Attention.
- Return a common response on a Read Attention to the terminal operator.
- Return a response and request additional input that can be read into the initial input area or into a new area.
- Request new I/O without requiring a return to the task to request additional input.

When the Read Attention exit routine has completed its execution, control is returned to the application program at the address specified in the DFHTC TYPE=READ macro instruction. The return is made whenever one of the following occurs:

- The exit routine issues no more requests for input.

- The exit routine issues a DFHTC TYPE=READ macro instruction and the operator terminates the input with a carriage return. (If the operator terminates the input with an Attention, the exit routine is reentered and is free to issue another READ.)

Note that terminal operator pressing of the Attention key on a read is recognized as a read attention only if read attention support is generated and the application programmer has included the RDATT=symbolic address operand in the DFHTC macro instruction requesting the input. If one or both of these conditions do not exist, the Attention key is treated as a normal read completion, that is, as if the Return key had been pressed.

Write Break (CICS/OS/VS)

If the terminal operator presses the Attention key on the 2741 while a message is being received, it is recognized as a Write Break if:

- Write Break support is generated into the system (available only in CICS/OS/VS).

- The write was initiated by a DFHTC TYPE=WRITE,WRBRK=symbolic address macro instruction (which has an implied WAIT).

When this occurs, the remaining portion of the message is not sent to the terminal. The write is terminated as though it were successful, and a new-line character (X'15') is sent to cause a carrier return. Control is returned to the application program at the address specified in the DFHTC TYPE=WRITE macro instruction.

If the Attention key is pressed and the Write Break feature is generated in CICS/OS/VS but the DFHTC TYPE=WRITE macro instruction does not have the WRBRK=symbolic address operand, the write break is treated as an I/O error. The same is true if the Attention key is pressed but the Write Break feature is not generated in CICS/OS/VS. A write is interruptible only if both conditions identified above are satisfied.

Note: TYPE=WAIT and/or SAVE can be coded with READ and/or WRITE, but only RDATT or WRBRK (not both) can be specified in one DFHTC macro instruction.

3270 PRINT FUNCTION

The 3270 Print Request facility allows either the application program or the terminal operator to request a printout of data currently displayed on the screen of a 3277 or 3275. One or both of these capabilities can be generated into the CICS/VS system through system generation macro instructions specified by the system programmer (see the discussions of the terminal control program and DFHTCT TYPE=TERMINAL macro instruction in the CICS/VS System Programmer's Reference Manual).

When these options are generated into CICS/VS, the terminal operator invokes the facility by pressing a specified Program Attention key (PA1, PA2, or PA3). The application program requests the facility by means of a DFHTC TYPE=PRINT macro instruction (see "DFHTC Macro Instruction").

For a 3277, this macro instruction causes the data currently displayed on the screen of the 3277 to be printed on the first available print-request-eligible printer (3284 or 3286) on the same 3271 or 3272 control unit. For a printer to be available, it must be in service

and not currently attached to a task. For a printer to be eligible, it must be on the same control unit, have a buffer capacity equal to or larger than the 3277, and FEATURE=PRINT must be specified for it by the system programmer.

For a 3275, this macro instruction causes the data currently in the 3275 display buffer to be printed on the 3284 Model 3 printer attached to the 3275.

If an eligible printer is not available when a DFHTC TYPE=PRINT macro instruction is executed, or if an error occurs during data transfer, the data in the display buffer is captured. The CICS/VS terminal abnormal condition program (DFHTACP) notifies the CICS/VS master terminal operator of the condition and passes control to a user-written terminal error program, which the user installation must provide to take action when this situation occurs.

TELETYPEWRITER PROGRAMMING CONSIDERATIONS

The teletypewriter (World Trade only) uses two different control characters for print formatting:

< carriage return, (X'22' in ITA2 code or X'15' in EBCDIC)

= line feed, (X'28' in ITA2 code or X'25' in EBCDIC)

The application programmer should always use < first; that is <= or <===, but never =<, otherwise following characters (data) may be printed while the typebar basket is moving to the left.

Message Format

Message Begin: To start a message on a new line at the left margin, always begin the message text with X'1517' (EBCDIC). CICS/VS, in this case, recognizes the X'17' and changes it to X'25' (X'17' is an IDLE character).

Message Body: To write several lines with a single transmission, separate the lines by X'1525', or if multiple blank lines are required, separate by X'152525...25'.

Message End before Next Input: To allow input of the next message on a line at the left margin, the preceding message should end with X'1517'. CICS/VS recognizes X'15' and changes the following character to X'25'.

Message End before Next Output: In case of two or more successive output messages, the message begin and the message end look the same, that is X'1517', except for the last message (see above). To make the message end of the preceding message distinguishable from the message begin of the next message, the next to the last character of the message end must not be X'15'.

Message Length

As a recommended practice for teletypewriter terminals, a message should not exceed a length of about 3000 bytes or approximately 300 words.

ASYNCHRONOUS TRANSACTION PROCESSING

Typically, a task to be run under CICS/VS is initiated from a terminal and processed at regular intervals until completion, according to system service patterns established for CICS/VS. This mode of operation is sometimes referred to as synchronous transaction processing, because the task has complete control of the terminal which initiated it.

Support for asynchronous transaction processing can also be generated into a CICS/VS system. This capability is designed primarily to permit a type of batch processing within CICS/VS. A task is initiated from a terminal as described above, but the specified transaction code causes a CICS/VS-provided asynchronous transaction processing program to read the data to an intrapartition data set (see "Transient Data Services"). In effect, data collection from a device such as the 2780 Data Transmission Terminal is possible. When the data has been read, the device is freed for other activity. An application program processes the data, and, upon operator request, output is queued for subsequent transmission to a specified terminal. If the automatic task-initiation feature is generated into CICS/VS, that application program can be initiated automatically when a specified trigger level is reached (that is, when a specified number of inputs have been entered in the intrapartition data set).

The asynchronous transaction processing (ATP) facility is designed specifically for handling input from batch terminals like the 2780 and 2770. Generally, ATP can also be used for other, interactive terminals like the 2741. However, ATP is not intended for, and will not support, input from the 3270, 2980, or 3735. Another consideration is that application programs intended to execute under control of ATP must not contain basic mapping support (BMS) macro instructions requesting BMS terminal paging facilities.

Additional information concerning the creation of user exits for asynchronous transaction processing and the coding of the exit routines is given in the CICS/VS System Programmer's Reference Manual. The initiation of ATP by means of terminal commands is described in the CICS/VS Terminal Operator's Guide.

DATA BASE CONSIDERATIONS

Use of segmented records, indirect accessing, duplicates data sets, the user's record identification field, and special considerations for DAM data sets are discussed below.

SEGMENTED RECORDS

An optional feature of CICS/VS file management allows the user to create and define a data set containing segmented records. A segmented record is one in which the components of the record have been identified (symbolically) and then grouped according to some logical relationship such as function or frequency of use.

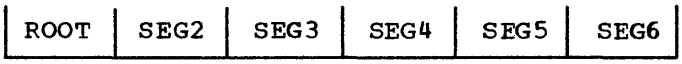
The identifiable groups are called segments. A segment is one or more adjacent fields within a record. Some segments appear in all records (for example, segments containing identification or major record control fields), while other segments apply to, and appear in, only certain records. If the application programmer plans to use segmented records in a program, the structure and individual segments of the data set must be defined in the file control table by the system programmer.

Segmented records offer numerous advantages. Having organized and defined the segments of a data set, the user can group them into segment sets and retrieve any set (or group) of segments by symbolically identifying that set. Since an individual segment can be a member of any number of segment sets, the user gains a high degree of flexibility in the retrieval process. Because only a part (a segment set) of a logical record is requested on a read-only operation, CICS/VS can extract only the requested segments, pass them to the application program, and free the main storage required for the entire logical record or block at the earliest possible time.

A saving in DASD space can be realized when segmenting is used with variable-length record format, since CICS/VS file management always compresses (packs) a segmented record before writing it to direct access storage. The space normally required for missing segments is thus eliminated, as are any slack bytes created when aligning segments in main storage.

With fixed-length records, compression does not save DASD space but it causes unused space to be consolidated at the ends of records. For example:

- Logical record as defined in the file control table



- Logical record as it appears on DASD with missing segments

The following general rules apply to the use of segmented records:

1. Segmented records can be used with VSAM, ISAM, or DAM data sets.
2. Segmented records can be used with any record format (that is, fixed, fixed blocked, variable, undefined) but are primarily advantageous when processing variable-length records.
3. A data set that contains segmented records cannot be an index data set in an indirect accessing hierarchy. The two CICS/VS features are mutually exclusive for any one data set. However, the primary (target) data set in an indirect accessing hierarchy may contain segmented records.
4. Every segment that may appear in a record, whether or not it actually exists in a particular record, must be defined in the file control table.

Segmented Record Formats

The user must describe all segments within a logical record of a segmented data set. As with any other data set, each segmented data set is first described in the file control table. That is, its basic characteristics (for example, block size, logical record length, and key length) must be described so that CICS/VS file management can physically access it. As an addendum to this basic data set description, the user must describe the segmented structure of the data set.

Every segment that may exist in a logical record of a segmented data set must be defined. While every logical record need not contain every segment, every logical record must contain at least the root (control) segment.

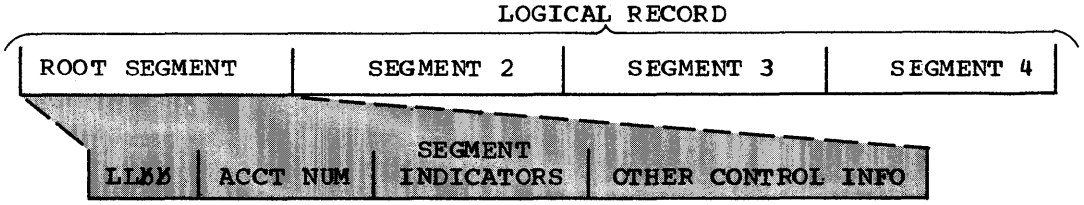
The root segment is a uniquely defined segment that must appear at the beginning of each logical record. It contains as a minimum:

1. The length of the record, if variable-length records are being used. This is a fullword (four bytes) of the form LLØØ, where LL is the record length and ØØ are two bytes reserved for system use.

2. Segment indicators, which indicate the presence or absence of each segment in the record as discussed in greater detail below.

In addition, the root segment may contain any other information essential to processing of the record by the user (for example, a major control field such as an account number).

The following is an example of a segmented record and the root (control) segment.



The sequence of the segments within a logical record must be fixed. That is, a segment cannot change position in relation to other segments of the record. Each segment can be fixed or variable in length. If the segment is variable in length, then the first byte must contain the length, in binary, of the segment, not including the length byte. Thus the maximum data length of a variable-length segment is 254 adjacent bytes; the maximum data length of a fixed-length record is 255 adjacent bytes. The number of bytes in a fixed-length segment or the maximum length of a variable-length segment is supplied to CICS/VS file management as part of the segment definition in the file control table.

Each segment has its own characteristics and these can differ from the characteristics of other segments. The length of each segment can differ from the lengths of other segments, and, if the segment is defined as variable-length, can change as a result of an update. Segments may be added or deleted; CICS/VS file management compresses and expands the record accordingly.

CICS/VS file management permits the user to specify the alignment requirements of each segment when that segment is brought into main

storage. This alignment may be on a one-byte, two-byte, four-byte, or eight-byte boundary. (The default alignment is on a one-byte boundary.) When the segmented record is written to direct access storage, any residual space (slack bytes) caused by alignment is eliminated by CICS/VS file management through the compress (packing) function.

Segment Indicators

Segment indicators are the means by which CICS/VS file management and the application program specify, and determine, the presence or absence of specific segments within a logical record. Two types of indicators are available to the user: bit-type segment indicators and displacement-type segment indicators. The application programmer must choose one of these types and define data sets accordingly. Regardless of the type of indicator, the following general rules apply to the use of segment indicators in processing a segmented record:

1. Segment indicators are always located in contiguous bytes within the root (control) segment. Every logical record contains a root segment and that root segment is always a part of any segment set brought into main storage. Therefore, the segment indicators are always accessible to the application program.

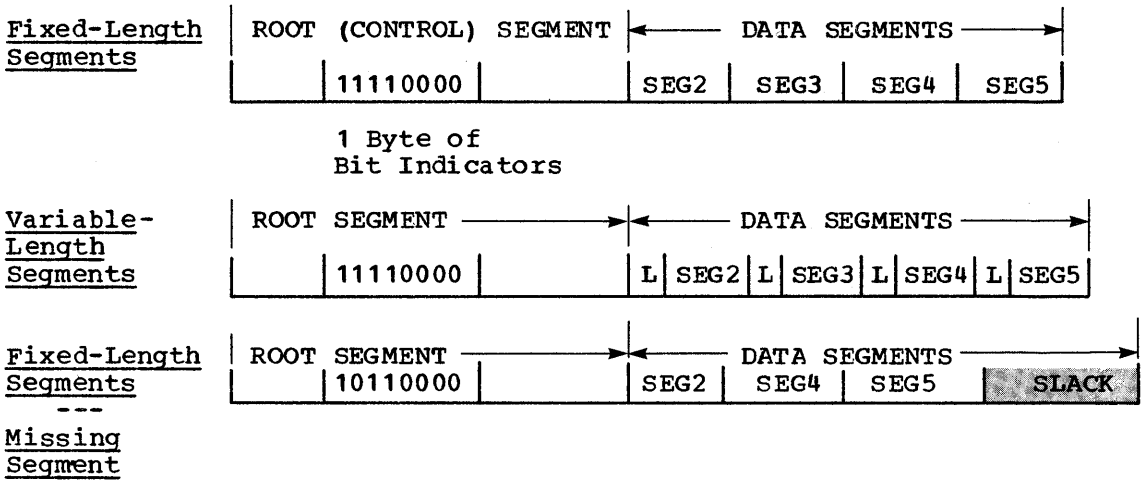
2. The location of the indicators within the root segment is defined in the file control table as some displacement from the beginning of the root segment.

3. There must be one indicator for each segment other than the root segment. The position of an indicator determines which segment it represents. Since the root segment does not require an indicator, the first indicator represents the first segment following the root (segment 2), the second indicator represents the second segment following the root (segment 3), and so on.

4. When retrieving segment sets, the application program must test appropriate indicators to determine whether requested segments are present. The program should never assume that a segment is present because it was requested as part of a segment set.

5. When adding or deleting segments from a record, the application program must set or reset appropriate indicators in the root segment of the record to reflect the change.

BIT-TYPE SEGMENT INDICATORS: With the bit-type indicator, each segment is represented by a bit position in the segment indicator field. One byte of indicators must be provided within the root segment for each eight segments in the logical record. If a given bit indicator is on (binary 1), the corresponding segment is present in the logical record. If a given bit indicator is off (binary 0), the corresponding segment is absent from the logical record. The following are examples of bit-type segment indicators:



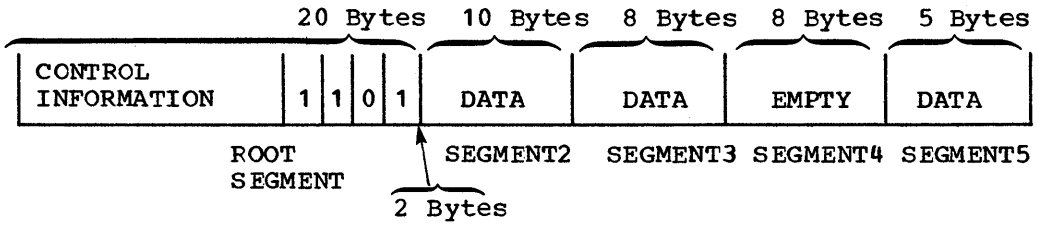
DISPLACEMENT-TYPE SEGMENT INDICATORS: With the displacement-type indicator, each segment is represented by one halfword (aligned) in the segment indicator field. In any given halfword indicator, a value of zero indicates that the corresponding segment is absent from the logical record. A nonzero value (binary) in any given halfword indicates that the corresponding segment is present and represents the displacement of the segment from the beginning of the logical record when the segments are packed.

Any displacement value placed in the halfword indicators when building a new record or adding and deleting segments from an existing record may be modified by CICS/VS file management when it compresses (packs) the segments before writing the record to direct access storage. Whenever CICS/VS packs segmented records, it places the displacement value of each segment into the corresponding halfword indicator (if displacement-type indicators are being used). However, CICS/VS file management does not change these displacement values when unpacking segmented records or when extracting selected segments of a segment set.

The application program should not rely on displacement values when accessing segments retrieved in a segment set; they should be used only as zero/nonzero indicators tc determine whether or not a requested segment is present. (See "Main Storage Processing of Segmented Records.")

The following example illustrates the basic concepts and considerations when using displacement-type segment indicators.

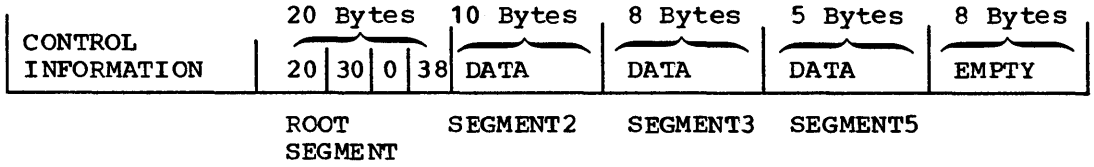
1. The following segmented record to be added to a segmented data set has been built in main storage by the user:



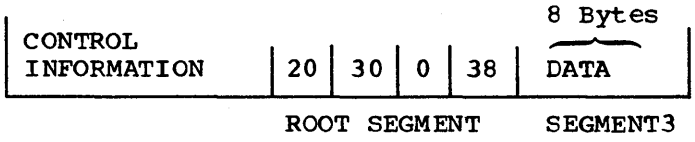
The user has placed data in three of the four defined segments and indicated their presence by placing a ncnzero value in the

corresponding halfword displacement indicators. Any nonzero value may be used (the 1 is only an example).

2. Before writing the record to the direct access data set, CICS/VS file management compresses the segments and modifies the displacement indicators. The above record appears as follows before it is written to DASD:



3. When a segment set is retrieved from the above record, the root segment is included as part of the segment set without modification. If the application program were to request a segment set (consisting of the root segment and segment 3) from the above record, the data might appear as follows:



Main Storage Processing of Segmented Records

When a segment set is requested from a segmented data set, the data is placed into a file work area (FWA). CICS/VS file management automatically calculates the amount of storage required and acquires the FWA through CICS/VS storage management. A CICS/VS-provided symbolic storage definition (DFHFWADS) can be used in conjunction with a user-defined layout to map the FWA.

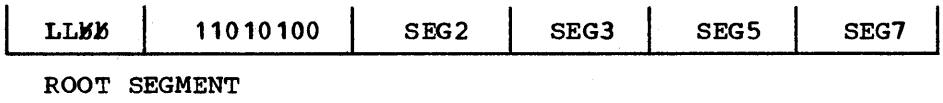
The FWA consists of control fields (used by CICS/VS management functions) and a data area into which the requested segments are placed by file management. The format of the retrieved segments within the data portion of the FWA is fixed; space is provided in the FWA and alignment requirements are met for each segment in the requested segment set, even though a segment may be missing. (For variable-length segments, the maximum space is provided.) As noted earlier, the application program must test appropriate segment indicators to determine whether specific segments are present in a requested segment set of a particular record. An update request on a segmented data set causes CICS/VS file management to use the universal segment set ALL when retrieving the record.

The following illustrations will help clarify the various considerations discussed thus far concerning main storage processing of segmented records.

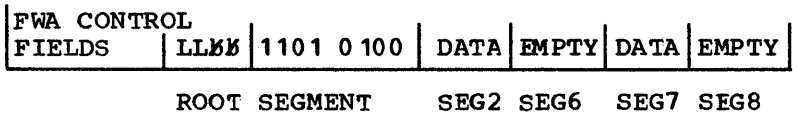
1. Logical record as defined in the file contrcl table:

ROOTSEG SEG2 SEG3 SEG4 SEG5 SEG6 SEG7 SEG8 SEG9

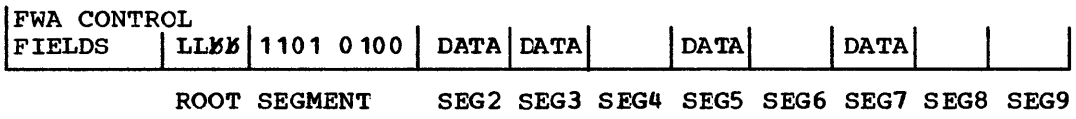
2. Logical record as it appears on DASD (assume variable-length records and bit-type segment indicators):



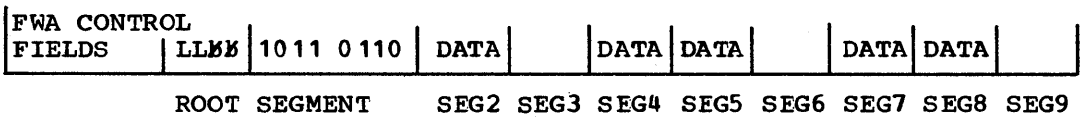
3. Logical record as it appears in the FWA after retrieval of a segment set (read-only) which includes the root segment, followed by segments 2, 6, 7, and 8.



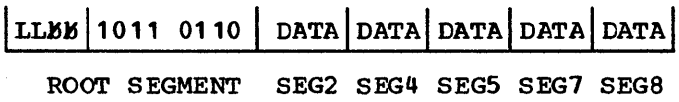
4. Logical record as it appears in the FWA after a retrieval for update (SEGSET=ALL):



5. Logical record as it appears in the FWA after the application program has added segments 4 and 8 and deleted segment 3. The indicators have been adjusted by the application program to reflect the change.



6. Logical record as it appears on DASD after packing:



Segment Sets

Once each segment has been defined (name and attributes specified), the user can specify as many segment sets as he desires. A segment set is a grouping of the root segment and at least one or more individual segments. Like the individual segments, the segment set is given a symbolic name which is used by the application program when processing a segmented data set. Any retrieval from a segmented data set is always by segment set.

Assume a logical record in a segmented data set has been defined as containing the following segments:

```
ROOTSEG
SEGMENT2
SEGMENT3
SEGMENT4
SEGMENT5
SEGMENT6
```

The user might wish to define the following segment sets:

Segment Set Name	Segments
SEGSETA	ROOTSEG
	SEGMENT2
	SEGMENT4
SEGSETB	ROOTSEG
	SEGMENT3
	SEGMENT4
	SEGMENT5

Whenever a segmented data set is defined in the file control table, a universal segment set is automatically generated which includes all segments defined for that data set. The symbolic identification of this universal segment set is ALL, and is automatically used by CICS/VS file management whenever the application program requests a "read for update" from a segmented data set. In other words, an update operation on a segmented data set always causes all segments to be presented to the application program, regardless of the segment set specified.

INDIRECT ACCESSING

Indirect accessing, an optional data base facility in CICS/VS, provides for the use of cross-index data sets to access another data set. The data set that is accessed by an index data set is known as the primary or target data set. This facility allows the user to furnish the search argument for an index data set along with the identification of the primary data set. CICS/VS, utilizing the user-defined index structure, carries out the search, involving as many levels (index data sets) as defined by the user, and ultimately retrieves the primary data set.

The following general rules apply to the indirect accessing facility:

1. A primary data set can have any number of index data sets. This is useful when multiple cross references to a master record exist.

2. Any data set can be both an index and a primary data set. The logical record content of any data base data set is user-defined and constructed, and therefore may contain certain master record information as well as a search argument for another data set.

3. There is no logical limit to the number of index levels (data sets) that the user may define in an index hierarchy. For example, data set A is an index to data set B, which is an index to data set C, and so on.

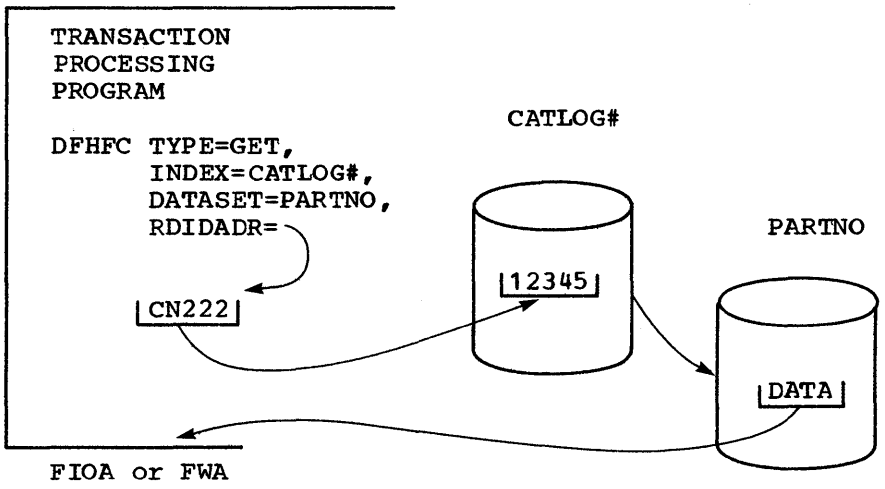
4. An index hierarchy can be any combination of VSAM, ISAM, and DAM data sets.

5. An index data set cannot contain segmented records. The two CICS/VS services are mutually exclusive for any one data set. However, a primary data set, which an index data set accesses, can have segmented records if it is not defined also as an index data set.

6. An index data set cannot reference more than one primary data set unless the index data set is multiply defined in the file control table.

7. If the index data set is a BDAM data set, it cannot be defined as blocked. However, the primary data set can be defined as blocked BDAM.

The following example shows a simple two-level index hierarchy for indirect accessing. The retrieval search begins with the index data set CATLOG#. The primary data set being accessed (and from which data is to be returned to the application program) is PARTNO. The search argument to be used in accessing the index data set (CATLOG#) is CN222. The contents of the record located by the search of the index data set (CATLOG#) contains the search argument for the next data set (12345 for search of PARTNO). The primary data set (PARTNO) is searched and the data record returned to the requesting program.



An installation must create and maintain all data sets in its data base, and define all data sets (both index and primary) in the file control table. Each data set, whether index and/or primary, is first described as a primary data set. That is, its basic physical characteristics (BLKSIZE, LRECL, KEYLEN, and so on) are defined so that CICS/VS file management can access it. If the data set is to be used as an index data set, the following information must also be specified:

1. The primary data set for which this data set is an index.

2. The location of the search argument, within the logical record of this data set, to be used for accessing the primary data set (or the next index data set).

If the user creates and defines an index hierarchy for indirect accessing, CICS/VS file management services any request requiring use of that hierarchy, provided the requesting application program adheres to the following general rules and considerations:

1. The symbolic name of the first index data set to be searched in the retrieval process must be specified through the INDEX operand of the DFHFC macro instruction. This data set can be any index data set in a hierarchy of indexes, not necessarily the highest level index data set. It can also be the primary data set being accessed without the use of an index data set. However, in the latter case, the DATASET operand must be used instead of the INDEX operand.

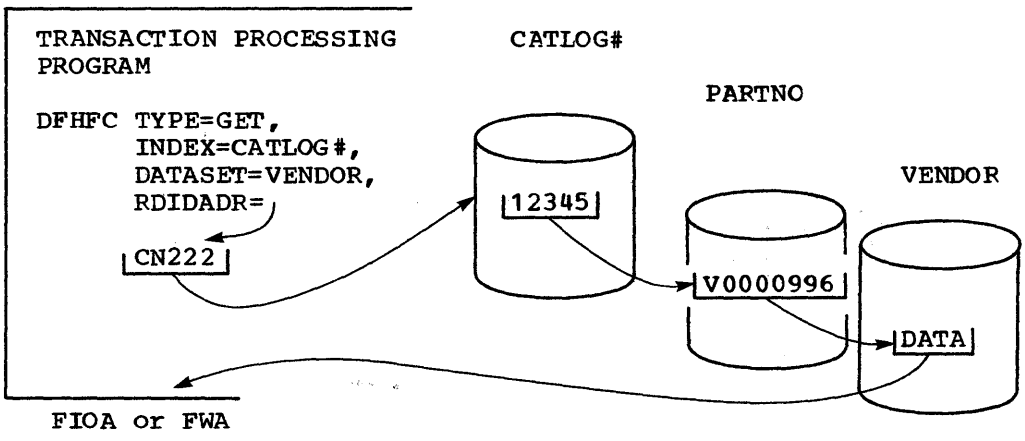
2. The symbolic name of the primary data set from which data is to be ultimately retrieved and returned to the requesting program must be specified through the DATASET operand of the DFHFC macro

instruction. Any number of intervening data sets can be used in the search; however, the user specifies only the first and the last data set. The user can limit a search to only a portion of an index hierarchy; that is, it is not necessary to search an entire index hierarchy, because the user can specify that the search begin at other than the highest-level index. Indexing levels cannot be omitted from within the hierarchical chain.

3. The search argument to be used by CICS/VS file management to access the first referenced data set must be specified through the RDIDADR operand of the DFHFC macro instruction. This operand points to a record identification field containing a VSAM key or relative byte address, an ISAM key, or DAM block reference information. (See "File Services" for additional details concerning search arguments that can be specified for file processing.) If multiple levels of index data sets are involved, CICS/VS file management acquires a search argument for the next data set from the logical record of each successive data set.

When stepping through a series of index data sets, CICS/VS file management uses the requesting prcgram's record identification field (specified in the RDIDADR operand) to store the search argument for each successive data set to be searched. The application programmer must ensure that this field is as large as the largest search argument that may be used in any given retrieval operation.

The following is an example of the above consideration in a three-level index hierarchy for indirect accessing. The search argument provided by the processing program is used to access the first index data set (CATLOG#) that provides the search argument for a second index data set (PARTNO) that provides the search argument for the primary data set (VENDOR) from which the data record is retrieved and returned to the application program. Since the search argument retrieved from the second index data set (PARTNO) is eight bytes in length (V0000996), the record identification field (RDIDADR) must be at least eight bytes in length, even though it initially contains only the five-byte search argument (CN222) for the first index data set.



DUPLICATE RECORDS

An optional feature of the indirect accessing approach to data base retrieval is the capability to indicate that a search argument in an index data set, which normally refers to the primary data set, instead refers to a "duplicates" data set. The need for or use of duplicates data sets may best be described as follows.

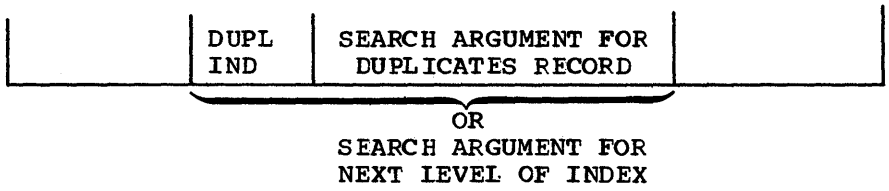
Assume that the application program requires access to an index data set organized by street address to obtain the name of the occupant at that address. The occupant's name is then used to access a primary data set organized by name.

For single occupancy, no problem exists. However, if multiple occupancy is possible, the index data set cannot directly equate a street address to a primary data set record. In this case, the search argument field in the index record must indicate that multiple occupants (duplicates) exist and that the search argument refers to a duplicates data set rather than the primary data set.

CICS/VS file management retrieves the referenced record from the duplicates data set and returns it to the application program with a response code indicating a duplicate record. The duplicate record may contain further information, which the application program can use to more accurately retrieve the appropriate record from the primary data set.

If an index data set is to indicate that there can be duplicate keys for entries in the primary data set to which it refers, this information must also be noted in the file control table entry which describes the index data set. The index data set record must contain a unique one-byte duplicates indicator (user-defined) in the first byte of the search argument field. Care must be taken to ensure that this indicator is a unique code; it cannot be the same as the first byte of a normal search argument for the primary data set.

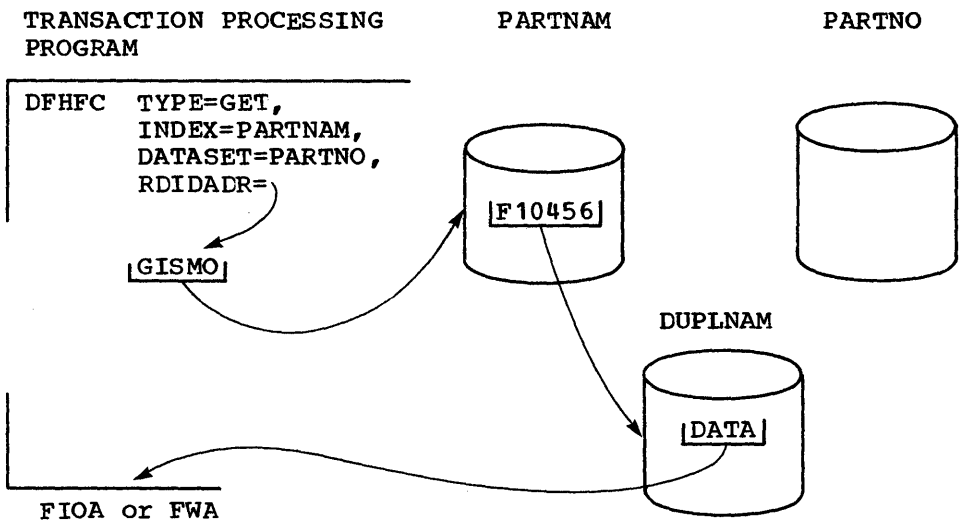
The rest of the search argument field contains the search argument used by CICS/VS file management to retrieve a record from the duplicates data set. This record may contain user-defined and user-constructed information that the application program can use to select the appropriate primary data set record. The following is an example of a search argument field in an index record that reflects duplicates:



The search argument for the duplicates data set must meet the same search argument format requirements as a normal cross-index data set. The length of the search argument used to access a duplicates data set is one byte smaller than a normal search argument because of the duplicates indicator.

The following is an example of an index hierarchy that includes a duplicates data set. The application program begins the retrieval by accessing the index data set (PARTNAM) and ultimately accesses the primary data set (PARTNO). The search argument (GISMO) provided by the application program is a valid one for the index data set (PARTNAM), but it provides a record containing a duplicates flag. When the duplicates indicator is detected, CICS/VS file management uses the new search argument (from the PARTNAM data set) to access the duplicates data set (DUPLNAM), returning the duplicates record to the application program.

In this example, the part name (GISMO) is not unique since there are several types of GISMOs in the part number (PARTNO) data set. The requesting program must provide qualifying data that indicates which GISMO is desired.



The record retrieved from the duplicates data set in the example might appear as follows:

```
GISMO   LARGE    9123     MED     9872     SMALL     9944
PARTNAM DESC     PARTNO   DESC    PARTNO   DESC      PARTNO
```

The application program might formulate a message to be routed to the inquiring terminal asking the terminal operator to make a choice. For example:

```
PART NAME REQUESTED HAS MULTIPLE ENTRIES

PLEASE SELECT SPECIFIC PART NUMBER

    PART NAME    DESCRIP      PART NUMBER

      GISMO      LARGE            9123

                 MED              9872

                 SMALL            9944
```

Once the terminal operator has made a selection, the program can make a direct retrieval from the primary (PARTNO) data set.

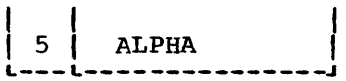
If the index record in the above example had not contained a duplicates indicator, CICS/VS file management would have used the search argument to access the primary data set (PARTNO) and retrieve the requested data.

RECORD IDENTIFICATION FIELD

The record identification field is used by the application program to communicate to CICS/VS file control the identity of a specific record required in an input/output operation. This field is identified by the RDIDADR operand of the DFHFC macro instruction. If multiple browse operations are performed concurrently by a single application program, a unique record identification field must exist for each operation. It is the application programmer's responsibility to provide the storage area for the record identification field. Generally, this storage is within the transaction work area (TWA) portion of the task control area (TCA), or some area acquired dynamically by the application program. It is not advisable to set up the record identification field within the application program.

For an ISAM data set, the record identification field is relatively simple in structure. It contains the key of the logical record.

For a VSAM data set, the record identification field contains either the logical record key or the relative byte address of the desired record. If the generic key option is used, the first byte of the field must contain the key length, in binary, and the remainder of the field must contain the generic key. For example:



The application programmer should understand that a partial key may be used as a search argument in a browse operation referring to either an ISAM or a VSAM data set. The ISAM partial key is an implied generic key, recognized as such because of padding with binary zeros in insignificant positions of the key. In contrast, a VSAM generic key is defined to be a generic key, with its length explicitly specified in the first byte of the record identification field. The ISAM implied generic key applies only to browse operations. The VSAM-defined generic key can be specified in numerous DFHFC macro instructions.

For a DAM data set, the record identification field structure is somewhat complex. The application program must supply the block reference information, physical key (if keyed data sets are being used), and the deblocking argument (if blocked data sets are being used).

The record identification field for DAM data sets is really a concatenation of three subfields, identified as follows:

1. Block reference

   The physical identifier for the data set in the DAM block is specified by the RELTYPE operand of the file control table and may be one of the following:

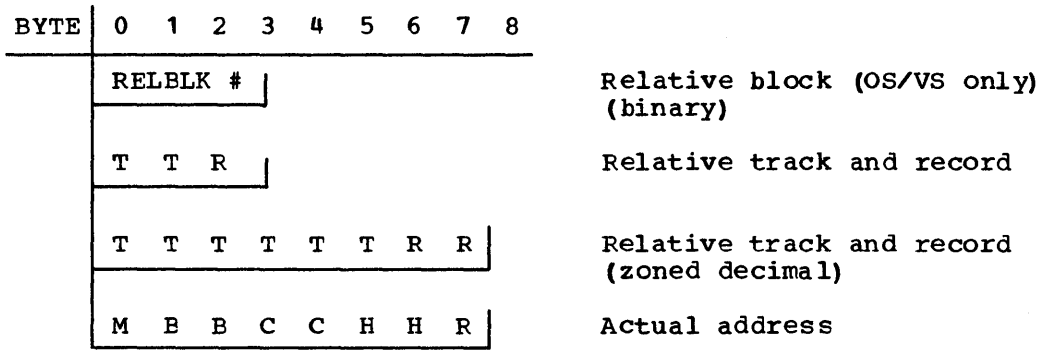
   a. Relative block (CICS/OS/VS only) three-byte binary (RELTYPE=BLK)

   b. Relative track and record - two-byte TT, one-byte R (RELTYPE=HEX)

   c. Relative track and record (zoned decimal format) six-byte TTTTTT, two-byte RR (RELTYPE=DEC)

   d. Actual address - eight-byte MBBCCHHR (RELTYPE omitted)

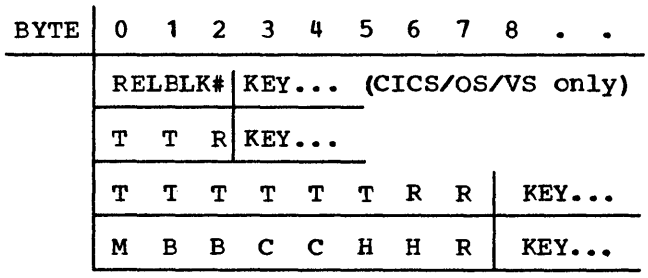
EXAMPLE



2. Physical key

The physical key is required only if the data set being accessed is written with recorded keys. This key must be the same length as specified in the BLKKEYL operand for the file control table entry which defines the data set. It must immediately follow the block reference information, which can be any of the above.

EXAMPLE

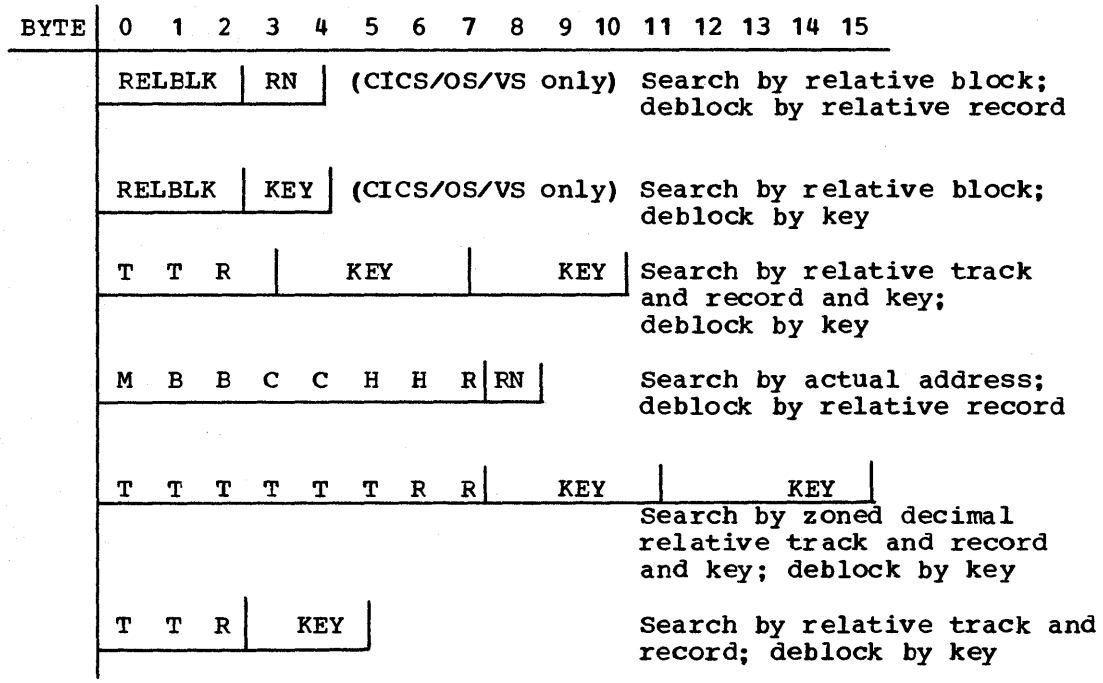


3. Deblocking argument

The deblocking argument is required only if the data set contains blocked records and specific logical records are to be retrieved from within a block. It is not mandatory that every physical record of a blocked data set be deblocked. If the application programmer does not specify a deblocking argument, an entire block is read into an FIOA. The deblocking argument may be either a key or a relative record number. The user's choice is specified in the RETMETH (retrieval method) operand of the DFHFC macro instruction. If present, the deblocking argument must immediately follow the physical key (if present) or the block reference (if the physical key is not present).

If the deblocking argument is a key, it must be the same length as specified in the KEYLEN operand of the file control table entry which describes the data set. The key used for deblocking need not be the same size as the physical record key (BLKKEYL). If the deblocking argument is a relative record number, it is represented by a one-byte binary number, with a value of zero representing the first logical record of a block.

EXAMPLE (physical key = 6 bytes, deblocking key = 3 bytes)



UPDATING NONKEYED DAM DATA SETS

Records in a nonkeyed DAM data set may be updated using either of two methods. One method is to issue a DFHFC TYPE=GET,TYPOPER=UPDATE to read the record, change the data in the FWA, and issue a DFHFC TYPE=PUT to physically update the record. This is the normal way that records are updated and should be used when portions of the record are to be changed and the actual contents of the record are unknown.

An alternative method may be used when the contents of the record to be updated are known, or when the entire record is to be changed, regardless of its contents. A DFHFC TYPE=GETAREA macro instruction is used to acquire an FWA, the record is built in the FWA, and a DFHFC TYPE=PUT,TYPOPER=UPDATE is issued to write the data at the location specified in the record identification field, destroying whatever was previously recorded at that location. This approach requires that both DAM update and DAM add capabilities be generated into the CICS/VS file control program (see the CICS/VS System Programmer's Manual).

ADDING RECORDS TO DAM DATA SETS

When adding new records to DAM data sets, the application programmer should be aware of the following considerations and restrictions:

1. When adding undefined or variable-length records (keyed or nonkeyed), the application programmer must indicate the track on which each new record is to be added. If space is available on the track, the new record is written following the last previously written record, and the record number is placed in the "R" portion of the record identification field of the record. The track specification may be in any of the acceptable formats except relative block. If zoned decimal relative format is used, the record number is returned as a two-byte zoned decimal number in the seventh and eighth positions of the record identification field.

In the CICS/DOS/VS system, an attempt to add a variable-length or undefined record is limited to the single track specified by the application programmer. If insufficient space is available on that track, a "no space available" error is returned, and the application programmer may then try to add the record on another track. Under these circumstances, the record is returned to the application program in an FWA, the address of which is at TCAFCAA. The programmer need only modify the track identification and issue another DFHFC TYPE=PUT,TYPOPER=NEWREC macro instruction to add the record on another track.

In the CICS/OS/VS system, the extended search option allows the record to be added to another track if no space is available on the specified track. Under these circumstances, the location at which the record was added is returned to the application program.

2. The addition of keyed fixed-length records to DAM data sets requires that the data set first be formatted with dummy records or "slots" into which new records may be added. (A dummy record is signified by a key of hexadecimal FFs; in CICS/OS/VS, the first byte of data contains the record-number.)

3. For nonkeyed, fixed-length records, the exact physical block reference must be given in the record identification field. The data in the new records is written in the exact location specified, destroying the previous contents of that location.

4. For keyed, fixed-length record additions, only the track information is used as a starting location for the search of a dummy key and record. When a dummy key and record are found, the new key and record replace it. The exact location at which the new record is located is returned to the application program in the block reference subfield of the record identification field.

   For example, suppose a user wishes to add a keyed, fixed-length record to a DAM data set. First, some algorithm determines that the search is to start at relative track 3. The record identification field of the new record might appear as follows:

   0 3 0 ALPHA

   T T R KEY

   When control is returned to the application program, the record identification field might reflect the fact that the record was added on relative track 4, record 6.

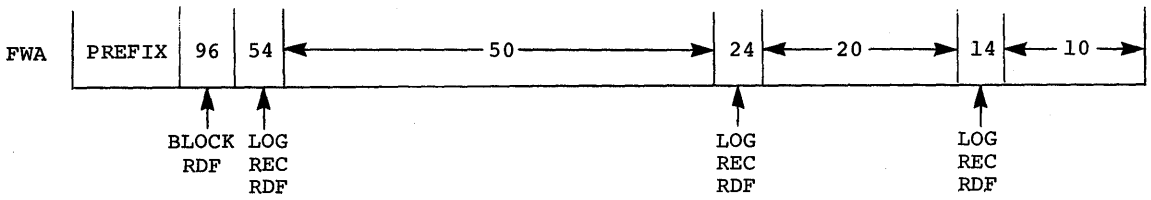
   0 4 6 ALPHA

   T T R KEY

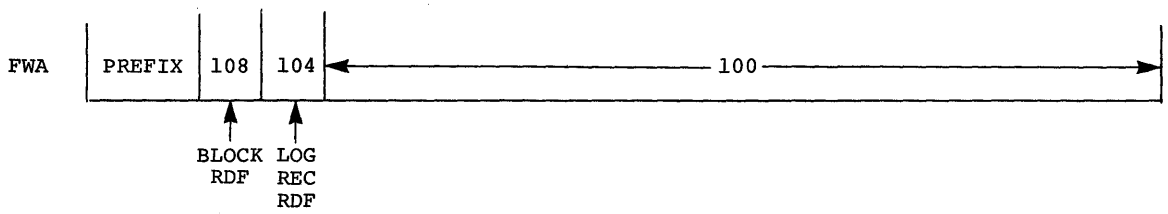
5. When adding records of undefined length, the length of the physical record must be placed in two-byte binary format at TCAFCURL. When an undefined record is retrieved, the application program must determine its length.

6. When making additions to a BDAM data set containing variable-length blocked or unblocked records, the application programmer must include a block record descriptor field (RDF) which contains the length (LLØØ) of the entire block to be written. For each logical record within that block, an RDF must be included which contains the length of the logical record.

Effectively, this allows the application to add a block containing multiple logical records. For example:



If only a single logical record is to be added, the block RDF is still required. For example:



BLOCK RDF = BLOCK RECORD DESCRIPTION FIELD
LOG REC RDF = LOGICAL RECORD RECORD DESCRIPTION FIELD

CHAPTER 12. REQUESTING DATA LANGUAGE/I SERVICE

The CICS/OS/VS application programmer can request Data Language/I (DL/I) services under CICS/OS/VS through a CALL written according to DL/I specifications or by issuing a DFHFC macro instruction. In response to such a request, control is passed to a CICS/VS-DL/I interface routine that acts as an interface between the CICS/OS/VS application program and the DL/I request handler in a DL/I task (which is an OS/VS subtask of CICS/OS/VS). This routine performs validity checks on the CALL list, sets up DL/I to handle this particular request, and passes control and the CALL list to DL/I. When the CICS/VS-DL/I interface routine regains control, it returns to the calling program unless a DL/I pseudo-ABEND has occurred, in which case the CICS/OS/VS transaction (task) is abnormally terminated.

Online access to DL/I data bases is also available under CICS/DOS/VS. The CICS/DOS/VS application programmer can request DL/I DOS/VS services through CALLs written according to DL/I DOS/VS specifications.

The CICS/VS-DL/I interface is capable of concurrently processing more than one DL/I request. This multithread capability enables the CICS/VS user to utilize DL/I in a more efficient manner and thus increase performance.

DL/I OS/VS and DL/I DOS/VS CALLs are similar in format and function. Therefore, the discussion of CALL formats and related considerations in this chapter applies to both CICS/OS/VS and CICS/DOS/VS.

The CICS/DOS/VS application programmer cannot issue CICS/VS DFHFC macro instructions to access a DL/I DOS/VS data base. Therefore, the following discussion of use of the DFHFC macro instruction to access DL/I data bases applies only to CICS/OS/VS.

QUASI-REENTRANT CONSIDERATIONS WITH REGARD TO DL/I (CICS/OS/VS)

Under CICS/VS, two or more transactions (tasks) may require the same application program at any given time during system operation. To avoid having to load multiple copies of the same program into main storage, CICS/VS application programs are required to be quasi-reentrant (see "Quasi-Reentrance"). Therefore, DL/I areas that may be modified under CICS/OS/VS (such as PCB pointers, I/O work areas, and segment search arguments) should not be placed in either static storage or working storage. Storage for such data must be obtained from CICS/VS dynamic storage by each transaction using the program.

Four steps must be performed by an application program requesting DL/I data base services. These steps are listed below and explained in the paragraphs that follow.

1. Obtain addresses of program communication blocks (PCBs) used by the application program.
2. Acquire storage for segment search arguments (SSAs) if they are to be used in the CALL.
3. Acquire I/O work areas for DL/I segments processed by the program.
4. Issue the DL/I CALL.

OBTAINING ADDRESSES OF PROGRAM COMMUNICATION BLOCKS

An application program that uses the CICS/VS-DL/I interface accesses data bases by means of program communication blocks (PCBs). One PCB for each data base is included in the program specification block (PSB) for the program. To successfully process DL/I CALLs within a CICS/VS transaction, the PSB for the transaction must be scheduled and the PCB addresses obtained before any DL/I CALLs are made. If they are not obtained, an INVREQ (invalid request) indicator is returned in response to any DL/I CALL within the program. For CICS/OS/VS, the scheduling process gives the transaction exclusive control of the PSB. This prevents other transactions from updating segment types that this transaction is updating. For CICS/DOS/VS, the DL/I interface gives control to the application program through "Intent Scheduling." For a further discussion of this process, consult the DL/I DOS/VS Application Programming Reference Manual.

A transaction may schedule only one PSB at a time. An attempt to schedule a second PSB while one is still scheduled causes the INVREQ indicator to be returned.

A sync point request (refer to the DFHSP macro instruction in Chapter 7) by a task that is scheduled to use DL/I resources implies the release of those resources. This means that if, after issuing a DFHSP TYPE=USER macro instruction, access to a DL/I data base is required, the desired PSB must be rescheduled through the DFHFC TYPE=(DL/I,PSB) macro instruction. The previous position of that data base has been lost.

DFHFC MACRO INSTRUCTION (CICS/OS/VS)

To schedule the desired PSB and obtain PCB addresses, the CICS/OS/VS application programmer uses a special form of the DFHFC macro instruction:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHFC │ TYPE=(DL/I,PCB)                                     │
│      │      │ [,PSB={'psbname'      }]                             │
│      │      │       {symbolic address}                            │
│      │      │       {YES            }                             │
│      │      │ [,NORESP=symbolic address]                          │
│      │      │ [,DLINA=symbolic address]                           │
│      │      │ [,PSBSCH=symbolic address]                          │
│      │      │ [,PSBNF=symbolic address]                           │
│      │      │ [,PSBFAIL=symbolic address]                         │
│      │      │ [,INVREQ=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=(DL/I,PCB)
indicates that PCB addresses are to be acquired.

Note: DL/I in the TYPE= operand may also be coded as DLI or DL1.

PSB=
specifies the name of the PSB to be scheduled for the transaction.

'psbname'
is the name of the PSB to be used.

symbolic address
    is the symbolic address of an eight-byte field containing the name of the PSB, padded to the right with blanks.

YES
    indicates that the name of the PSB has been placed in TCADLPSB by the application program.

If this operand is omitted, the name of the program associated with the transaction in the CICS/VS program control table (PCT) is used as the PSB name.

NORESP=symbolic address
specifies the entry label of the user-written routine to which control is passed if the PSB is located and the PCB addresses are returned. NORESP signifies "normal response." If this operand is omitted, but a described condition applies, processing continues with the next sequential instruction in the application program.

DLINA=symbolic address
specifies the entry label of the user-written routine to which control is passed if the CICS/VS-DL/I interface is inactive.

PSBSCH=symbolic address
specifies the entry label of the user-written routine to which control is passed if a PSB is already scheduled for this task.

PSBNF=symbolic address
specifies the entry label of the user-written routine to which control is passed if the PSB cannot be found in the PSB directory.

PSBFAIL=symbolic address
specifies the entry label of the user-written routine to which control is passed if the PSB fails to initialize.

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is passed if: (1) a DLINA, PSBSCH, PSBNF, or PSBFAIL condition occurs and the associated operand is omitted, or (2) some other condition making the request invalid is detected.

If an INVREQ condition occurs and the INVREQ and an associated expansion operand(s) are both omitted, processing continues with the next sequential instruction in the application program.

If the PSB has been located, TCADLPCB contains the address of a list of PCB addresses in the sequence in which the PCB addresses were specified during the PSBGEN of this PSB. If the PSB cannot be found, TCADLPCB contains zero. If the PSB pool or DMB pool is too small to hold the requested blocks even when no other PSBs or DMBs are in their pools, the transaction is terminated with the ADLA ABEND code.

DL/I CALL STATEMENT (CICS/DOS/VS)

Upon receiving control from CICS/DOS/VS, a CICS/DOS/VS application program must issue a DL/I call to perform scheduling before attempting to access DL/I data bases. If the scheduling call is successful, the address of the PCB list is returned in the field TCADLPCB and TCAFCTR is set to zeros. If the call is unsuccessful, TCADLPCB contains zeros and TCAFCTR contains a one-byte return code. Before continuing with subsequent DL/I calls, it is the application programmer's responsibility to test these indicators to determine whether scheduling was successful.

The general format of the CALL statement to request scheduling is as follows:

For Assembler language:

```
CALLDLI  ASMTDLI ,  ([parmcount,]function,psb)
         CBLTDLI
```

For ANS COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]function,psb.
```

For PL/I:

```
CALL PLITDLI ([parmcount,]function,psb);
```

where:

parmcount
: is the name of a binary fullword containing the parameter count (value of one or two). This parameter is optional.

function
: is the name of the field containing the four-character function 'PCBØ'.

psb
: is the name of the field containing the one- to seven-character PSB generation name which the application program accesses. This parameter is optional. If it is omitted, the PSB name is assumed to be the first PSB name associated with the application program name in the DL/I application control table generation.

For a more detailed discussion of the CALLDLI macro instruction, consult the DL/I DOS/VS Application Programming Reference Manual.

BUILDING SEGMENT SEARCH ARGUMENTS

Both CICS/OS/VS and CICS/DOS/VS application programmers can use segment search arguments (SSAs) in a DL/I CALL to identify a specific segment, or, if qualified, to identify the range of values within which a segment exists. In addition, the CICS/OS/VS programmer can specify SSAs in a DFHFC TYPE=DL/I macro instruction. If used, SSAs must be built by the application programmer before a DL/I CALL is issued. (CICS/OS/VS application programmers should see the IMS/VS Application Programming Reference Manual for information concerning how to build an SSA; CICS/DOS/VS users should refer to the DL/I DOS/VS Application Programming Reference Manual.)

In a DL/I application program, SSAs are built in fixed storage within the program. In a CICS/VS application program, SSAs must be built in dynamic storage to maintain the quasi-reentrance of the program.

The storage acquired to build the SSAs is addressed as follows:

- For Assembler-language programs, the address should be placed in the register that establishes addressability for the SSA dynamic storage.
- For ANS COBOL programs, the address is moved to the BLL pointer for this storage. The BLL pointer is defined under the COPY DFHBLLDS statement in the Linkage Section and must be in the same relative position in the BLL list as the 01 statement for the SSA dynamic storage is among the 01 statements in the Linkage Section.

- For PL/I, the address is stored in the variable upon which the SSA dynamic storage is based.

After the storage has been acquired and the SSAs built, DL/I CALLs in which the SSAs are used can be issued by the application program. The names of the SSAs to be used, if any, are specified in the parameter list of the CALL. Under CICS/OS/VS, a DFHFC TYPE=DL/I macro instruction can also be used. In a DFHFC TYPE=DL/I macro instruction, the application programmer can specify the number and names of the SSAs in different ways:

1. SSAS=NO indicates that there are no SSAs in this CALL.

2. SSAS=(ssacount,ssa1,ssa2,...), where ssacount is optional, represents either the fixed-point number of SSAs in the CALL or the symbolic address of the fullword that contains the number of SSAs. Specifying a field to contain the number of SSAs provides the application programmer with flexibility in writing one DFHFC statement to be used in many different CALLs. ssa1, ssa2,..., are the symbolic names of the SSAs.

3. SSALIST=YES indicates that the application programmer has built a list of fullwords, cptionally containing the number of SSAs (which may be zero) in the first word, and the addresses of the SSAs in the following words, and that he has stored the address of this list at TCADLSSA.

4. SSALIST=symbolic address indicates that the address of an SSA list built by the application programmer as indicated in item 3 is at the location specified.

In Assembler-language programs, ssacount,ssa1,ssa2,..., can be contained in registers if the specifications are enclosed in parentheses.

ACQUIRING AN I/O WORK AREA

When issuing a request for DL/I services, the address of a work area, either that in which a current segment is contained or that in which DL/I is to place the segment in a retrieval CALL, is required. This area must be specified by the CICS/OS/VS or CICS/DOS/VS programmer who issues a DL/I CALL. It may be provided by the interface, if the programmer desires, for a retrieval-type DFHFC macro instruction.

If the CICS/OS/VS application programmer knows the address of the work area to be used in the DFHFC macro instruction, including the case for which storage is acquired for a retrieval-type (Gxxx) request, he specifies the name of the pointer to that storage in the WRKAREA=name operand, or he places the address of the storage in TCADLIO before issuing the request and specifies WRKAREA=YES.

If the application programmer wishes to allow the interface to obtain the work area for a retrieval-type request, he does not include the WRKAREA operand in the DFHFC macro request. If the request was serviced successfully, the address of an acquired I/O work area is in TCADLIO. The address at TCADLIO is the address of the storage accounting area (SAA) preceding the retrieved data. The area becomes the responsibility of the programmer and is not freed until he frees it or until the transaction terminates. If the application programmer elects to free the work area, he must use a DFHSC TYPE=FREEMAIN macro instruction.

Note: The address of the I/O area is specified as the address of the storage accounting area preceding the data when a DFHFC macro

instruction is used; the address of the first byte of the data area is required when a DL/I CALL is used.

ISSUING THE DL/I CALL

The application program request for DL/I services may be either a CICS/VS DFHFC macro instruction (CICS/OS/VS) or a DL/I call (CICS/OS/VS or CICS/DOS/VS).

DFHFC MACRO INSTRUCTION (CICS/OS/VS)

The general format of the DFHFC macro instruction to request that a particular DL/I function be performed is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │ DFHFC │ TYPE=(DL/I [,function])                             │
│      │      │ [,PCB={symbolic address}]                           │
│      │      │       {(register)      }                            │
│      │      │ [,WRKAREA={symbolic address}]                       │
│      │      │           {YES            }                         │
│      │      │           {(register)     }                         │
│      │      │ [,SSAS={NO                                    }]    │
│      │      │        {([ssacount][,ssa1][,ssa2,...])        }    │
│      │      │        {([(register1)][,(register2),...])     }    │
│      │      │ [,SSALIST={YES            }]                        │
│      │      │           {NO             }                         │
│      │      │           {symbolic address}                        │
│      │      │           {(register)     }                         │
│      │      │ [,NORESP=symbolic address]                          │
│      │      │ [,NOTOPEN=symbolic address]                         │
│      │      │ [,DLINA=symbolic address]                           │
│      │      │ [,FUNCNS=symbolic address]                          │
│      │      │ [,INVREQ=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=(DL/I [,function])
: specifies the two- to four-byte name of the function to be performed. If the function is not specified, it is assumed to be in TCADLFUN.

  Note: DL/I in the TYPE= operand may also be coded as DLI or DL1.

PCB=
: specifies the field that contains the address of the PCB.

  symbolic address
  : is the symbolic address of the field containing the address of the PCB.

  (register)
  : is valid only when Assembler language is used and is the number of a register that contains the address of the PCB.

WRKAREA=
: specifies the address of the work area to be used.

symbolic address
is the symbolic address of a field that contains a pointer to the work area.

YES
indicates that the address of the work area to be used has been placed in TCADLIO by the application program.

(register)
is valid only when Assembler language is used and is the number of a register that contains the address of the work area.

If this operand is omitted and a Gxxx function is to be performed, the CICS/VS-DL/I interface acquires storage for the work area and returns the address of the work area at TCADLIO. The application program must save this address upon return. If any other type of function is requested, the application program must provide the work area. (See programming note 1.)

SSAS=
indicates whether or not segment search arguments are used in this request and, if so, identifies them.

NO
indicates that no SSAs are used in this request.

([ssacount][,ssa1][,ssa2,...])
specifies the names of segment search arguments used in this request (thereby creating an SSA list). The ssacount parameter specifies the number of SSAs to be used; it is the address of a fullword containing the count, or, in the case of Assembler language, may be expressed as a numeric value. Each ssa specification represents an element of the SSA list (see programming note 2).

([(register1)][,(register2),...])
is interpreted as described above; that is, register1 contains a count of the SSAs in the list or is the first list entry, register 2 is the first or second list entry (depending on whether a count has been specified), and so on.

If this operand is specified, SSALIST cannot be specified.

SSALIST=
indicates whether or not segment search arguments are used in this request and if so, identifies the list containing these arguments.

YES
indicates that a list of segment search arguments is used and that the address of the list has been placed in TCADLSSA by the application program.

NO
indicates that no SSA list is used in this request.

symbolic address
is the symbolic address of a field that contains the address of the SSA list.

(register)
is valid only when Assembler language is used and is the number of a register that contains the address of the SSA list.

If this operand is specified, SSAS cannot be specified.

NORESP=symbolic address
specifies the entry label of the user-written routine to which control is passed when the application program regains control. The CICS/VS-DL/I interface must have been able to pass control to DL/I and a DL/I pseudo-ABEND of the transaction cannot have occurred. The user must check the return code in the PCB to determine whether DL/I was able to service the request. If this operand is omitted, control is passed to the next sequential instruction in the application program.

NOTOPEN=symbolic address
specifies the entry label of the user-written routine to which control is passed if this data base is logically (not necessarily physically) closed. The PCB does not contain a DL/I AI status code.

DLINA=symbolic address
specifies the entry label of the user-written routine to which control is passed if the CICS/VS-DL/I interface is inactive.

FUNCNS=symbolic address
specifies the entry label of the user-written routine to which control is passed if a DL/I functional request (a request other than PCB or TERM) is made and the task does not have a PSB scheduled.

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is passed if: (1) a DLINA or FUNCNS condition occurs and the associated operand is omitted, or (2) some other condition making the request invalid is detected. If an INVREQ condition occurs and the INVREQ and an associated expansion operand(s) are both omitted, processing continues with the next sequential instruction in the application program.

Programming Notes:

1. A work area whose address is specified in a DFHFC macro instruction or placed at TCADLIO prior to execution of the DFHFC macro instruction includes the CICS/VS storage accounting area prefix. A work area specified in a CALLDLI or CALL statement does not.

2. The first element of an SSA list, or it may point to a fullword containing this count; the remaining elements represent addresses of SSAs. If the first element of an SSA list is not a count, all elements of the SSA list are assumed to be addresses of SSAs; the high-order bit of the last element of the list must be set on to indicate the end of the list.

DL/I CALL STATEMENT (CICS/OS/VS OR CICS/DOS/VS)

DL/I data base services are available to CICS/VS application programs through CALL statements. The CALL statement formats for American National Standard (ANS) COBOL and PL/I are similar. For Assembler-language application programs, a CALLDLI macro instruction is used. The general formats of the DL/I calls are as follows:

For Assembler language:

```
CALLDLI  ASMTDLI [,([parmcount,]function,pcb,workarea[,ssa,...])]
         CBLTDLI
```

For ANS COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]function,pcb,workarea[,ssa,...].
```

For PL/I:

```
CALL PLITDLI (parmcount,function,pcb,workarea[,ssa,...]);
```

where:

parmcount
: is the name of a binary fullword containing the parameter count or argument count of the arguments which follow; this is optional for Assembler language and ANS COBOL.

function
: is the name of the field containing the four-character DL/I input/output CALL function desired.

pcb
: is the program communication block (PCB) name (or DSECT name if Assembler).

workarea
: is the name of the I/O work area.

ssa1 to ssan
: are the names of the segment search arguments (SSAs); these are optional.

Notes:

1. If no parameters are specified in an Assembler-language CALLDLI macro instruction, register 1 is assumed to contain the address of a parameter list.

2. In Assembler language, an alternative format may be used:

   ```
   CALLDLI  ASMTDLI ,MF=(E,(register) or address)
            CBLTDLI
   ```

   where:

   address is the address of the parameter list, or register contains the address of the parameter list.

RELEASING A PSB IN THE CICS/VS APPLICATION PROGRAM

DFHFC MACRO INSTRUCTION (CICS/OS/VS)

To reduce pool and intent contention, the CICS/OS/VS application program can release the PSB after a DL/I service has been requested.

It is recommended that conversational programs release the PSB before writing to a terminal so that other transactions can use the PSB while the conversational program is waiting for an operator response.

To ensure data-base integrity, a request to release a PSB implies the end of a logical unit of work for the entire task. This means that a DFHSP TYPE=USER is issued on behalf of a task that is releasing a PSB.

To release a PSB for use by other transactions, the CICS/OS/VS application programmer issues a macro instruction of the following format:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │DFHFC │ TYPE= (DL/I,{TERM})                                  │
│      │      │              {T   }                                  │
│      │      │ [,DLINA=symbolic address]                            │
│      │      │ [,TERMNS=symbolic address]                           │
│      │      │ [,INVREQ=symbolic address]                           │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=(DL/I,TERM)
specifies that the PSB is to be released for use by other transactions, or, if not required, its pool space and associated DMB pool space may be released for other purposes.

Notes:

1. DL/I in the TYPE= operand may also be coded as DLI or DLl.

2. TERM may be abbreviated as T.

DLINA=symbolic address
specifies the entry label of the user-written routine to which control is passed if the CICS/VS-DL/I interface is inactive.

TERMNS=symbolic address
specifies the entry label of the user-written routine to which control is passed if a termination request is made and the task has no PSB scheduled.

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is given if: (1) a DLINA or TERMNS condition occurs and the associated operand is omitted, or (2) some other condition making the request invalid is detected. If an INVREQ condition occurs and the INVREQ and an associated expansion operand(s) are both omitted, processing continues with the next sequential instruction in the application program.

Before issuing any other DL/I CALLs or DFHFC macro instructions requesting DL/I access to a data base, the application programmer must again issue a schedule-type DFHFC macro instruction. All positioning in data bases referred to by the transaction is lost when the PSB is released. Thus, if the program was processing a hierarchy through GNxx requests before releasing the PSB, it is necessary to explicitly reposition the data bases after issuing another schedule-type DFHFC macro instruction, to continue the GNxx requests.

DL/I CALL STATEMENT (CICS/DOS/VS)

If the CICS/DOS/VS application program desires to relinquish control of the PSB, it must issue a terminal call to DL/I. The general format of the CALL statement to request termination is as follows:

For Assembler language:

```
CALLDLI  ASMTDLI , ([parmcount,]function)
         CBLTDLI
```

For ANS COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]function.
```

For PL/I:

```
CALL PLITDLI ([parmcount,]function);
```

where:

parmcount
: is the name of a binary fullword containing the parameter count value of one.

function
: is the name of the field containing the four-character function 'TERM' or 'TØØØ'.

When a termination call is issued for a previously scheduled PSB, the resources acquired for the task are released, and tasks awaiting the resources are given an opportunity to be scheduled.

CHECKING THE RESPONSE TO A REQUEST FOR DL/I SERVICES

When the application programmer issues a request for CICS/VS-DL/I interface operations, he can check the reponse to his request to determine subsequent processing that should be carried out. One step in doing so is to specify the entry-point names (symbolic labels) of user-written exception-handling routines, any of which may be executed as a result of the check. He can do this in any of three ways:

1. Include the entry-point names in operands of the DFHFC macro instruction by which the service is requested.

2. Include the entry-point names in operands of a

```
DFHFC   TYPE=CHECK,                                          *
          .
          .
```

   macro instruction immediately following the DFHFC macro instruction by which the service is requested.

3. Include instructions immediately following the DFHFC macro instruction that examine the response code set automatically by CICS/VS and transfer control to an exception-handling routine accordingly.

When the third approach above is used, the application programmer must know the CICS/VS response codes and their meanings. If the Assembler-language or PL/I programmer elects to use this approach, he can access the response codes for NORESP, INVREQ, and NOTOPEN at TCAFCTR; the response codes for all other conditions can be accessed at TCADLTR. The ANS COBOL programmer can access the response codes for NORESP, INVREQ, and NOTOPEN at TCAFCRC; the response codes for all other conditions can be accessed at TCADLTR. The possible response codes and the conditions to which they correspond are identified in the right-hand column of Figure 12-1. CICS/VS-DL/I interface requests for which the conditions are applicable are shown at the left.

DL/I Interface Request by DFHFC Macro Instruction	Condition	Response Code		
		Assembler	ANS COBOL	PL/I
(DL/I,PCB),(DL/I [,function]),CHECK	NORESP (Normal Response)	X'00'	12-0-1-8-9 (FCNORESP)	00000000
(DL/I[,function]),CHECK	NOTOPEN (Not open)	X'0C'	12-4-8-9 (FCNOTOPEN)	00001100
All	INVREQ (Invalid Request)	X'08'	12-8-9 (FCINVREQ)	00001000
		INVREQ Expansion Codes at TCADLTR		
(DL/I,PCB),CHECK	PSBNF (PSB Not Found)	X'01'	12-1-9 (DLPSBNF)	00000001
CHECK	TASKNA (Task Not Authorized)	X'02'	12-2-9 (DLTASKNA)	00000010
(DL/I,PCB),CHECK	PSBSCH (PSB Already Scheduled	X'03'	12-3-9 (DLPSBSCH)	00000011
CHECK	LANGCON (Language Conflict)	X'04'	12-4-9 (DLLANGCON)	00000100
(DL/I,PCB),CHECK	PSBFAIL (PSB Initialization Failed)	X'05'	12-5-9 (DLPSBFAIL)	00000101
CHECK	PSBNA (PSB Not Authorized)	X'06'	12-6-9 (DLPSBNA)	00000110
(DL/I,T),CHECK	TERMNS (Termination Not Scheduled)	X'07'	12-7-9 (DLTERMNS)	00000111
(DL/I[,function]), CHECK	FUNCNS (Funtion Not Scheduled)	X'08'	12-8-9 (DLFUNCNS)	00001000
	DLINA (DL/I Not Active)	X'FF'	12-11-0-7-8-9 (DLINA)	11111111

Notes:

1. The TASKNA, LANGCON, and PSBNA conditions apply only to CICS/DOS/VS.

2. The names enclosed in parentheses in the ANS COBOL column indicate the 88-level definitions provided by CICS/VS. These names may be used in testing for the respective conditions in a COBOL program.

Figure 12-1. CICS/VS-DL/I Interface Response Codes

The operands that are appropriate for the forms of the DFHFC macro instruction used to request CICS/VS-DL/I interface processing are explained in the preceding discussion. A complete discussion of the DFHFC TYPE=CHECK macro instruction is given earlier under "Test Response to a Request for File Services." The operands that are appropriate for checking the CICS/VS-DL/I interface response and their meanings are summarized below:

```
┌──────┬──────┬──────────────────────────────────────────────────────┐
│      │      │                                                      │
│      │DFHFC │ TYPE=CHECK                                           │
│      │      │ [,NORESP=symbolic address]                           │
│      │      │ [,DLINA=symbolic address]                            │
│      │      │ [,PSBSCH=symbolic address]                           │
│      │      │ [,PSBNF=symbolic address]                            │
│      │      │ [,PSBFAIL=symbolic address]                          │
│      │      │ [,FUNCNS=symbolic address]                           │
│      │      │ [,TERMNS=symbolic address]                           │
│      │      │ [,LANGCON=symbolic address]      CICS/DOS/VS only    │
│      │      │ [,TASKNA=symbolic address]       CICS/DOS/VS only    │
│      │      │ [,PSBNA=symbolic address]        CICS/DOS/VS only    │
│      │      │ [,INVREQ=symbolic address]                           │
│      │      │ [,NOTOPEN=symbolic address]                          │
│      │      │                                                      │
└──────┴──────┴──────────────────────────────────────────────────────┘
```

where:

TYPE=CHECK
: indicates that the CICS/VS-DL/I interface response is to be checked.

NORESP=symbolic address
: specifies the entry label of a user-written routine to which control is passed upon normal execution of the request.

DLINA=symbolic address
: specifies the entry label of the user-written routine to which control is passed if the CICS/VS-DL/I interface is inactive.

PSBSCH=symbolic address
: specifies the entry label of the user-written routine to which control is passed if a PSB is already scheduled for this task.

PSBNF=symbolic address
: specifies the entry label of the user-written routine to which control is passed if the PSB cannot be found in the PSB directory.

PSBFAIL=symbolic address
: specifies the entry label of the user-written routine to which control is passed if the PSB fails to initialize.

FUNCNS=symbolic address
: specifies the entry label of the user-written routine to which control is passed if a DL/I function request (a request other than PCB or TERM) is made and the task has no PSB scheduled.

TERMNS=symbolic address
: specifies the entry label of the user-written routine to which control is passed if a termination request is made and the task has no PSB scheduled.

LANGCON=symbolic address (CICS/DOS/VS only)
specifies the entry label of the user-written routine to which control is passed if the calling program is in a different source language than the called PSB.

TASKNA=symbolic address (CICS/DOS/VS only)
specifies the entry label of the user-written routine to which control is passed if the calling task is not authorized to access DL/I data bases.

PSBNA=symbolic address (CICS/DOS/VS only)
specifies the entry label of the user-written routine to which control is passed if the task is not authorized to access this PSB.

INVREQ=symbolic address
specifies the entry label of the user-written routine to which control is passed if: (1) a DLINA, PSBSCH, PSBNF, PSBFAIL, FUNCNS, TERMNS, LANGCON, TASKNA, or PSBNA condition occurred and the associated operand was omitted, or (2) some other condition making the request invalid is detected. If an INVREQ condition occurs and the INVREQ and an associated expansion operand(s) are both omitted, processing continues with the next sequential instruction in the application program.

NOTOPEN=symbolic address
specifies the entry label of the user-written routine to which control is passed if the data base specified in the PCB used in this request is logically (not necessarily physically) closed. The PCB does not contain a DL/I AI status code.

The application programmer may use the DFHFC TYPE=CHECK macro instruction following a CALLDLI, CALL, or DFHFC TYPE=(DL/I[,function]) macro instruction. This macro instruction does not check the DL/I return codes in the PCB. If DL/I issues a pseudo-ABEND during processing of the request, control is not returned to the application program. The transaction is terminated with CICS/VS ABEND code ADLA. For CICS/DOS/VS, if DL/I issues a pseudo-ABEND during a call, the transaction is terminated with a Dnnn ABEND code where nnn is the DL/I pseudo-ABEND code.

If the application programmer does not provide for the checking of a particular response, and if the exception condition corresponding to that response occurs, program flow proceeds to the instruction following the DFHFC macro instruction in the application program.

DL/I REQUESTS IN AN ASSEMBLER-LANGUAGE PROGRAM (CICS/OS/VS)

The application programmer must first get the addresses of the PCB. When CICS/OS/VS returns from servicing the PCB request, if the programmer loads register 1 from TCADLPCB, his program is in the same state as after an ENTRY DLITCBL statement has been executed in an IMS/VS DL/I application program.

The examples that follow show the options available to the application programmer in a few of the acceptable combinations. Note that, the application program must be quasi-reentrant. Note also that, if a DFHFC macro instruction is issued, the PCB and WRKAREA operands are used to specify the addresses of pointers to fields rather than the addresses of fields desired.

```
        COPY DFHTCADS                   COPY TCA DEFINITION - INCLUDES
*                                       DL/I FIELDS
PSBNAME DC   CL8'PSBNAME'               NAME OF PSB TO BE USED
```

```
PCBFUN   DC    CL4'PCBb'                  PCB FUNCTION
PCBPTRS  DSECT                           PCB POINTERS RETURNED BY
*                                        INTERFACE
PCB1PTR  DS    F                         STORAGE FOR PCB POINTERS
PCB2PTR  DS    F
         .
         .
WORKAPTR DS    F                         STORAGE FOR POINTER IN I/O WORK
*                                        AREA
PCB1     DSECT                           PCB DSECT
         .
         .
         .
PCB2     DSECT                           PCB DSECT
         .
         .
         .
WRKAREA  DSECT                           DL/I WORK AREA DSECT
         DS    2F                        STORAGE PREFIX
WORKA1   DS    CL40                      WORK AREA
SSAREA   DSECT                           SSA DSECT
         DS    2F                        STORAGE PREFIX
SSA1     DS    CL40                      SSA1 LAYOUT
SSA2     DS    CL20                      SSA2 LAYOUT
         .
         .
         .
         DFHFC TYPE=(DL/I,PCB)           USE PSB FOR THIS PROGRAM
         DFHFC TYPE=(DL/I,PCB),          GET PCB'S IN 'PSB14'             *
               PSB='PSB14'
         DFHFC TYPE=(DL/I,PCB),          GET PCB'S IN SPECIFIED PSB       *
               PSB=psbname
         MVC TCADLPSB,=CL8'PSBA'         PUT PSB NAME IN TCA
         DFHFC TYPE=(DL/I,PCB),          GET PCB'S OF PSB NAMED IN TCA    *
               PSB=YES
         L     R1,TCADLPCB               GET ADDRESS OF PCB ADDR LIST
         USING PCBPTRS,R1                REG 1 IS BASE OF PCB POINTERS --
*                                        USER MUST PROVIDE ADDRESSABILITY
*                                        TO PCB'S WHEN USING THEM
* ISSUE A PCB REQUEST VIA CALLDLI
         CALLDLI CBLTDLI,(PCBFUN)        USE PSB FOR THIS PROGRAM
         CALLDLI CBLTDLI,(PCBFUN,PSBNAME)GET PCB'S IN SPECIFIED PSB
         L     R1,TCADLPCB               GET ADDRESS OF PCB ADDRESS LIST
* ACQUIRE STORAGE FOR WORK AREA
         DFHSC TYPE=GETMAIN,...          GET STORAGE FOR WORK AREA
         L     R2,TCASCSA                REG 2 IS BASE FOR WORK AREA
         USING WRKAREA,R2                TELL ASSEMBLER
* ACQUIRE STORAGE FOR SSA'S
         DFHSC TYPE=GETMAIN,...          GET STORAGE FOR SSA'S
         L     R3,TCASCSA                REG 3 IS BASE FOR SSA'S
         USING SSAREA,R3                 INDICATE TO ASSEMBLER
*
         CALLDLI CBLTDLI,(function,PCB1,WORKA1,SSA1,SSA2)
*
* CALL DL/I VIA DFHFC MACRO -- VARIOUS EXAMPLES
*
* EXAMPLE 1
*
         DFHFC TYPE=(DL/I,function),                                      *
               PCB=PCB1PTR,              PCB IS POINTED TO                *
               WRKAREA=WORKAPTR,         WORK AREA IS POINTED TO          *
               SSAS=(2,SSA1,SSA2),       SSA COUNT AND SSAS SPECIFIED     *
               NORESP=GOOD1              NORMAL RESPONSE BRANCH
*
* EXAMPLE 2
```

```
*
        MVC   TCADLPCB,PCB1PTR          PRELOAD PCB POINTER
        LA    R0,WRKAREA                PICK UP WORK AREA ADDRESS
        ST    R0,TCADLIO                STORE IN TCA
        DFHFC TYPE=(DL/I,DLET),         FUNCTION SPECIFIED              *
              WRKAREA=YES,              WORK AREA ADDRESS PRELOADED     *
              SSAS=NO                   NO SSAS
*
* EXAMPLE 3
*
        MVC   TCADLFUN,=CL4'GU'         PRELOAD FUNCTION
        DFHSC TYPE=GETMAIN,...          GET STORAGE FOR SSA LIST
        L     R4,TCASCSA                PICK UP STORAGE ADDRESS
        LA    R4,8(R4)                  BYPASS PREFIX
        LA    R0,1                      GET COUNT OF SSA'S
        ST    R0,0(R4)                  STORE IN SSA LIST
        LA    R0,SSA1                   GET ADDRESS OF 'SSA1'
        ST    R0,4(R4)                  STORE IN SSA LIST
        ST    R4,TCADLSSA               STORE LIST ADDRESS IN TCA
        OI    4(R4),X'80'               SET ON THE END-OF-LIST BIT
        DFHFC TYPE=DL/I,                DL/I CALL, FUNCTION PRELOADED   *
              PCB=PCB1PTR,              POINTER TO PCB TO BE USED       *
                                        INTERFACE WILL PROVIDE WORK AREA*
              SSALIST=YES               PROBLEM PROGRAM PROVIDES SSA LIST
        L     R3,TCADLIO                PICK UP ADDRESS OF SUPPLIED
*                                       WORK AREA
```

DL/I REQUESTS IN AN ANS COBOL PROGRAM (CICS/OS/VS)

Upon program entry, the ANS COBOL programmer should obtain PCB addresses by issuing a DFHFC TYPE=(DL/I,PCB) request. After CICS/OS/VS returns control, the programmer moves the contents of TCADLPCB to the BLL pointer which is the base for the layout of the PCB pointers in the Linkage Section. He then moves the addresses of the PCBs to their BLL pointers to provide the base addresses for the PCBs. When this is done, the program is in the same state as after an ENTRY 'DLITCBL' USING PCB1,PCB2 statement has been executed in an IMS/VS DL/I application program.

For an explanation of how BLL pointers to 01 statements in the Linkage Section are defined, see the discussion of ANS COBOL application programming in Chapter 4.

Examples showing how to write DL/I requests are given below. Only some combinations of operands are shown, but other combinations are acceptable. Note that, in a DFHFC request, BLL pointers to the PCB and work area are used rather than actual field names. This is the only way the addresses can be passed to DL/I.

```
WORKING-STORAGE SECTION.
77  PSBNAME PICTURE X(8) VALUE 'PSBNAME'.
77  PCB-FUNCTION PICTURE X(4) VALUE 'PCB '.
77  FUNCTION-1 PICTURE X(4) VALUE 'DLET'.
77  SSA-COUNT PICTURE 9(8) COMPUTATIONAL VALUE +2.
LINKAGE SECTION.
01  DFHBLLDS COPY DFHBLLDS
    02  ...                            NOTE POINTERS TO OTHER CICS/VS
*                                       AREAS NEEDED
    02  B-PCB-PTRS PICTURE 9(8) COMPUTATIONAL.
    02  B-PCB1 PICTURE 9(8) COMPUTATIONAL.
    02  B-PCB2 PICTURE 9(8) COMPUTATIONAL.
    02  B-WORKAREA PICTURE 9(8) COMPUTATIONAL.
    02  B-SSAS PICTURE 9(8) COMPUTATIONAL.
01  DFHCSADS COPY DFHCSADS.
```

```
01  DFHTCADS COPY DFHTCADS.
    .                                   NOTE TWO DEFINITIONS.
    .                                   NOTE OTHER AREA DEFINITIONS.
    .
01  PCB-PTRS.
    02  PCB1-PTR PICTURE 9(8) COMPUTATIONAL.
    02  PCB2-PTR PICTURE 9(8) COMPUTATIONAL.
01  PCB1.
    .
    .
    .
01  PCB2.
    .
    .
    .
01  WORKAREA.
    02  FILLER PICTURE X(8).           NOTE STORAGE PREFIX.
    02  WORKA1 PICTURE X(40).
    .
    .
    .
01  SSAREA.
    02  FILLER PICTURE X(8).
    02  SSA1 PICTURE X(40).
    02  SSA2 PICTURE X(60).
    .
    .
    .
PROCEDURE DIVISION.
* GET PCB ADDRESSES
    DFHFC TYPE=(DL/I,PCB)               GET PSB FOR THIS PROGRAM
* GET PCB ADDRESSES VIA CALL
    CALL 'CBLTDLI' USING PCB-FUNCTION,PSBNAME.
                                        NOTE GET PCB'S FOR SPECIFIED PSB.
* SAVE PCB ADDRESSES IN BLL TABLE SO PCB'S CAN BE ADDRESSED
    MOVE TCADLPCB TO B-PCB-PTRS.
    MOVE PCB1-PTR TO B-PCB1.
    MOVE PCB2-PTR TO B-PCB2.
* OPTIONALLY, ACQUIRE STORAGE FOR WORK AREA
    DFHSC TYPE=GETMAIN,...
    MOVE TCASCSA TO B-WORKAREA.
* OPTIONALLY, ACQUIRE STORAGE FOR SEGMENT SEARCH ARGUMENTS
    DFHSC TYPE=GETMAIN,...
    MOVE TCASCSA TO B-SSAS.
* CALL DL/I VIA CALL
    CALL 'CBLTDLI' USING FUNCTION-1,PCB1,WORKA1,SSA1,SSA2.
* EXAMPLE 1 OF DFHFC MACRO INSTRUCTION
    DFHFC TYPE=(DL/I,GHU),              FUNCTION                        *
          PCB=B-PCB1,                   PCB POINTER                     *
          WRKAREA=B-WORKAREA,           WORK AREA POINTER               *
          SSAS=(SSA-COUNT,SSA1,SSA2)    SSA COUNT AND NAMES
* EXAMPLE 2 OF DFHFC MACRO INSTRUCTION
    MOVE 'GNP ' TO TCADLFUN.            NOTE PRELOAD FUNCTION.
    MOVE B-PCB1 TO TCADLPCB.            NOTE PRELOAD PCB ADDRESS.
    DFHFC TYPE=DL/I,                    FUNCTION PRELOADED              *
          SSAS=NO                       PCB ADDRESS PRELOADED           *
                                        WORK AREA TO BE ACQUIRED        *
                                        NO SSA'S
    MOVE TCADLIO to B-WORKAREA.         NOTE SAVE ACQUIRED WORK AREA ADDR.
```

DL/I REQUESTS IN A PL/I PROGRAM (CICS/OS/VS)

Upon entry to his program, the PL/I application programmer should get PCB addresses through a DFHFC TYPE=(DL/I,PCB) statement. When CICS/VS returns, the base address of a structure of PCB pointers is in TCADLPCB. The PL/I programmer must move the value from TCADLPCB to the based variable for his declared structure of PCB pointers. He then loads the pointers to all PCBs from this structure. When this has been done, the program is in the same state as an IMS/VS DL/I application program in which the

```
DLITPLI: PROCEDURE (pctname1,...) OPTIONS (REENTRANT,MAIN);
```

statement has been executed.

The PL/I programmer may then make DL/I requests, either through CALLs or DL/I DFHFC macro instructions. Note that the PCB and WRKAREA operands in a DFHFC request specify the addresses of pointers to fields rather than of the fields desired.

```
%INCLUDE DFHCSADS;                              /* CSA DEFINITION */
%INCLUDE DFHTCADS;                              /* TCA DEFINITION - INCLUDES */
                                                /* DL/I FIELDS */
    DECLARE 1 PCB_POINTERS BASED (B_PCB_PTRS),
              2 PCB1_PTR POINTER,
              2 PCB2_PTR POINTER;
                .
                .
    DECLARE 1 PCB1 BASED (BPCB1),               /* PCB DEFINITIONS */
              2 ...
              2 ... ;
    DECLARE 1 PCB2 BASED (BPCB2),
              2 ...
              2 ... ;
    DECLARE 1 DLI_IOAREA BASED (BDLIIO), /* DL/I I/O AREA */
              2 STORAGE_PREFIX CHAR(8),
              2 IOKEY CHAR(6),
              2 ... ;
    DECLARE 1 DLI_SSADS BASED (BSSADS),  /* DL/I SSA LIST */
              2 STORAGE_PREFIX CHAR(8),
              2 SSA1,
                3 SSA1KEY CHAR(6),
                3 ...
              2 SSA2,
                3 ...
                3 ...;
    DECLARE PSBNAME CHAR(8) INIT ('PSBNAME');
    DECLARE PCB_FUNCTION CHAR(8) INIT ('PCB ');
/*  OBTAIN PCB POINTERS  */
       DFHFC TYPE=(DL/I,PCB)              GET PSB FOR THIS PROGRAM
/*  OBTAIN PCB POINTERS VIA CALL */
       CALL PLITDLI (PARM_CT,PCB_FUNCTION,PSBNAME): /* GET SPECIFIED PSB */
/*  SAVE POINTERS IN PCB BASES  */
    B_PCB_PTRS=TCADLPCB;
    BPCB1=PCB1_PTR;
    BPCB2=PCB2_PTR;
/*  ACQUIRE STORAGE FOR DL/I I/O AREA  */
       DFHSC TYPE=GETMAIN,CLASS=USER,...
       BDLIIO=TCASCSA;
/*  OPTIONALLY, ACQUIRE STORAGE IN WHICH TO BUILD SSA'S */
       DFHSC TYPE=GETMAIN,CLASS=USER,...
       BSSADS=TCASCSA;
/*  OPTIONALLY, BUILD SEGMENT SEARCH ARGUMENTS */
```

```
      SSA1KEY=TERMKEY;
           .
           .
           .
/*  CALL DL/I  */
      CALL PLITDLI(PARM_CT,DLI_FUNCTION,PCB1,IOKEY,SSA1,
      SSA2);
/* EXAMPLE 1 OF DFHFC MACRO INSTRUCTION */
    DFHFC TYPE=(DL/I,ISRT),                                          *
          PCB=BPCB1,                    PCB POINTER                  *
          WRKAREA=BDLIIO,               WORK AREA POINTER            *
          SSAS=(2,SSA1,SSA2)            SSA COUNT AND NAMES
/* EXAMPLE 2 OF DFHFC MACRO INSTRUCTION */
    TCADLPCB=BPCB1;
    DFHFC TYPE=(DL/I,GU),               PCB PRELOADED                *
          SSAS=(SSA_COUNT,SSA1,SSA2)    WORK AREA TO BE ACQUIRED     *
                                        SSA COUNT AND NAMES
    BDLIIO=TCADLIO;                     /* SAVE WORK AREA ADDRESS */
/* EXAMPLE 3 OF DFHFC MACRO INSTRUCTION */
    TCADLFUN='GN';                      /* PRELOAD FUNCTION */
    TCADLIO=BDLIIO;                     /* PRELOAD WORK AREA ADDRESS */
    DFHFC TYPE=DL/I,                    FUNCTION PRELOADED           *
          PCB=BPCB1,                    PCB POINTER                  *
          WRKAREA=YES,                  WORK AREA ADDRESS PRELOADED  *
          SSAS=NO                       NO SSA'S
```

When using the PL/I optimizer, all SSAs used in DFHFC calls and all parameters used in CALLs must be defined as elementary items. This can be done by defining structures based on the same pointers as the structures containing the nonelementary definitions.

```
   DECLARE 1 DLI_CALL_SSADS BASED (BSSADS),
             2 STORAGE_PREFIX CHAR(8),
             2 CALL_SSA1      CHAR(...),
             2 CALL_SSA2      CHAR(...);
/* SET UP SSA1 AND USE IN CALL */
      SSA1KEY=SEARCH_KEY;
      DFHFC TYPE=DL/I,
            SSAS=(SSA_COUNT,CALL_SSA1)
      CALL PLITDLI (PARM_CT,FUNCTION,PCB1,IOKEY,CALL_SSA1);
```

APPENDIX A. EXECUTABLE CICS/VS SAMPLE PROGRAMS

This appendix contains an executable applicatiCn program that performs a limited message switching function; that is, data collection, message entry, and message retrieval. The program is shown in each of the languages supported under CICS/VS: Assembler language, American National Standard (ANS) COBOL, and PL/I.

```
*********************************************************************
      A S S E M B L E R      E X A M P L E      P R O B L E M
*********************************************************************
*       TITLE 'CICS/VS MESSAGE SWITCHING PROGRAM EXAMPLE'           *
        DFHCOVER
*********************************************************************
* * * *              A P P L I C A T I O N   P R O G R A M    * * * *
*********************************************************************
* * *                  D U M M Y   S E C T I O N S              * * *
*********************************************************************
        COPY  DFHCSADS                  COPY COMMON SYSTEM AREA DSECT
        EJECT                           LISTING CONTROL CARD - EJECT
        COPY  DFHTCADS                  COPY TASK CONTROL AREA DSECT
TWATSRL DS    H                         TEMPORARY STORAGE RECORD LENGTH
        DS    H
TWATDDI DS    CL4                       DESTINATION IDENTIFICATION
TWAREAI DS    CL4                       RETRIEVE ALL INDICATOR
TWAQEMCI DS   C                         QUEUE EMPTY MESSAGE CONTROL IND
        EJECT                           LISTING CONTROL CARD - EJECT
TCTTEAR EQU   11                        TERM CONT TABLE TERM ENT ADR RG
        COPY  DFHTCTTE                  COPY TERM CONT TABLE TERM ENTRY
TIOABAR EQU   10                        TERM I/O AREA BASE ADDR REG
        COPY  DFHTIOA                   COPY TERMINAL I/O AREA DSECT
TIOADATA DS   0CL80                     DATA AREA
TIOATID DS    CL4                       TRANSACTION IDENTIFICATION
        DS    C                         DELIMITER
TIOARRI DS    0CL6                      RESUME REQUEST IDENTIFICATION
TIOARAI1 DS   0CL3                      RETRIEVE ALL INDICATOR 1
TIOADID DS    CL4                       DESTINATION IDENTIFICATION
TIOASSF DS    0CL4                      SUSPEND STORAGE FACILITY IDENT
        DS    C                         DELIMITER
TIOAMBA DS    0C                        TERMINAL MESSAGE BEGINNING ADDR
TIOARAI2 DS   CL3                       RETRIEVE ALL INDICATOR 2
*********************************************************************
        SPACE 8                         LISTING CONTROL CARD - SPACE 8
TDIABAR EQU   9                         TRANS DATA IN AREA BASE ADDR RG
        COPY  DFHTDIA                   COPY TRANS DATA INPUT AREA
        EJECT                           LISTING CONTROL CARD - EJECT
*********************************************************************
* * * *              A P P L I C A T I O N   P R O G R A M    * * * *
*********************************************************************
CICSATP CSECT                           CONTROL SECTION - APPL TEST PGM
        USING *,3                       USING REGISTER 3 AT *
        LR    03,14                     LOAD PROGRAM BASE REGISTER
        B     ATPIPIN                   GO TO INIT PROG INSTR ENTRY
*********************************************************************
        EJECT                           LISTING CONTROL CARD - EJECT
*********************************************************************
* * *                  D E C L A R A T I V E S                  * * *
*********************************************************************
MCPDIEM DC    Y(MCPDEML-4)              TERMINAL MESSAGE LENGTH
        DC    H'0'
```

```
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARDCOPY TERM IDLE CHARACTERS
         DC    C'DESTINATION IDENTIFICATION ERROR - PLEASE RESUBMIT'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
MCPDEML  EQU   *-MCPDIEM                TERMINAL MESSAGE TOTAL LENGTH
*********************************************************************
*********************************************************************
*                      D A T A   C O L L E C T I O N                *
*********************************************************************
DCPDCAML DC    Y(L'DCPDCAMD)            DATA COLL ACKNOWLEDGMENT LEN
         DC    H'0'
DCPDCAMD DC    C' DATA COLLECTION HAS BEEN REQUESTED AND IS ABOUT TO BE*
               GIN  '                   DATA COLLECTION ACKNOWLEDGMENT
DCPEODML DC    Y(L'DCPEODMD)            END OF DATA MESSAGE LENGTH
         DC    H'0'
DCPEODMD DC    C' THE DATA HAS BEEN RECEIVED AND DISPATCHED TO THE DESI*
               GNATED DESTINATION  '    END OF DATA MESSAGE
DCPEOVML DC    Y(L'DCPEOVMD)
         DC    H'0'
DCPEOVMD DC    C' END OF VOLUME REQUEST HAS BEEN RECEIVED  '
DCPSRAM  DC    Y(DCPSRAL-4)             TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARD COPY TERM IDLE CHARACTERS
         DC    C'DATA COLLECTION SUSPENSION HAS BEEN REQUESTED'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
DCPSRAL  EQU   *-DCPSRAM                TERMINAL MESSAGE TOTAL LENGTH
DCPRRAM  DC    Y(DCPRRAL-4)             TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARDCOPY TERM IDLE CHARACTERS
         DC    C'DATA COLLECTION RESUMPTION HAS BEEN REQUESTED AND IS '
         DC    C'ABOUT TO BEGIN'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
DCPRRAL  EQU   *-DCPRRAM                TERMINAL MESSAGE TOTAL LENGTH
*********************************************************************
         SPACE 4                        LISTING CONTROL CARD - SPACE 4
*********************************************************************
*                        M E S S A G E   E N T R Y                  *
*********************************************************************
MEPMEAML DC    Y(L'MEPMEAMD)            MSG ENTRY ACKNOWLEDGMENT LNGTH
         DC    H'0'
MEPMEAMD DC    C' YOUR MESSAGE HAS BEEN RECEIVED AND DISPATCHED TO THE *
               DESIGNATED DESTINATION  ' MESSAGE ENTRY ACKNOWLEDGMENT
*********************************************************************
         SPACE 4                        LISTING CONTROL CARD - SPACE 4
*********************************************************************
*                      M E S S A G E   R E T R I E V A L            *
*********************************************************************
MRPNMMM  DC    Y(MRPNMML-4)             TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARDCOPY TERM IDLE CHARACTERS
         DC    C'THERE ARE NO MORE '
         DC    C'MESSAGES QUEUED FOR THIS DESTINATION'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
MRPNMML  EQU   *-MRPNMMM                TERMINAL MESSAGE TOTAL LENGTH
MRPNMQM  DC    Y(MRPNQML-4)             TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARDCOPY TERM IDLE CHARACTERS
         DC    C'THERE ARE NO MESSAGES QUEUED FOR THIS DESTINATION'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
MRPNQML  EQU   *-MRPNMQM                TERMINAL MESSAGE TOTAL LENGTH
*********************************************************************
```

```
         EJECT                          LISTING CONTROL CARD - EJECT
*****************************************************************
* * *                  I M P E R A T I V E S                  * * *
*****************************************************************
* *                                                             * *
*****************************************************************
         DS    0D                       STORAGE ALIGNMENT - DOUBLEWORD
         DC    CL32'MESSAGE CONTROI PROGRAM'
ATPIPIN  DS    0D                       INITIAL PROGRAM INSTRUCTION ENT
         L     TCTTEAR,TCAFCAAA         LOAD TERM CONT AREA ADDR REG
         L     TIOABAR,TCTTEDA          LOAD TERM I/O AREA ADDR REG
         CLC   =C'DSDC',TIOATID         COMPARE TRANSACTION IDENT
         BE    ALPDCPN                  GO TO DATA COLLECTION PROG IF =
         CLC   =C'DSME',TIOATID         COMPARE TRANSACTION IDENT
         BE    ALPMEPN                  GO TO MESSAGE ENTRY PROG IF =
         CLC   =C'DSMR',TIOATID         COMPARE TRANSACTION IDENT
         BE    ALPMRPN                  GO TO MESSAGE RETRIEVAL PROG
         DFHPC TYPE=ABEND,                                            *
               ABCODE=XAPT
         EJECT                          LISTING CONTROL CARD - EJECT
*****************************************************************
* *                    A P P L I C A T I O N   L O G I C        * *
*****************************************************************
* *                      D A T A   C O L L E C T I O N          * *
*****************************************************************
         DC    CL32'DATA COLLECTION PROGRAM'
*****************************************************************
ALPDCPN  DS    0H                       DATA COLLECTION PROGRAM ENTRY
         CLC   =C'RESUME',TIOARRI       COMPARE FOR RESUME REQUEST
         BNE   DCPRRBN                  GO TO RESUME REQUEST BYPASS
         MVC   TIOATDL(DCPRRAL),DCPRRAM MOVE TERMINAL MESSAGE TO OUTPUT
         MVC   TCATSDI(4),=C'DSDC'      MOVE TEMP STRG DATA IDENT
         MVC   TCATSDI+4(4),TCTTETI     MOVE TEMP STRG DATA IDENT
         DFHTS TYPE=GET,                                              *
               TSDADDR=TWATSRL,                                       *
               NORESP=DCPRRNR,                                        *
               RELEASE=YES
         DFHPC TYPE=ABEND,                                            *
               ABCODE=XDCR
DCPFEOV  EQU   *                        FORCED END OF VOLUME ROUTINE
         DFHTD TYPE=FEOV                ISSUE TRANSIENT DATA MACRO
         MVC   TIOATDL((4+L'DCPEOVMD)),DCPEOVML
         DFHTC TYPE=(WRITE)
         B     RETURN
*****************************************************************
DCPRRBN  EQU   *                        RESUME REQUEST BYPASS ENTRY
         MVC   TWATDDI,TIOADID          MOVE DESTINATION IDENTIFICATION
         MVC   TCATDDI,TWATDDI
         CLC   TIOAMBA(4),=C'FEOV'      CHECK FOR FORCED END OF VOL REQ
         BE    DCPFEOV                  BRANCH TO END OF VOLUME ROUTINE
         MVC   TIOATDL((4+L'DCPDCAMD)),DCPDCAML
DCPRRNR  EQU   *                        RESUME REQUEST NORMAL RESPONSE
         DFHTC TYPE=(WRITE)
         DFHTC TYPE=(READ)
*****************************************************************
         SPACE 4                        LISTING CONTROL CARD - SPACE 4
DCPTEWN  DS    OH                       TERMINAL EVENT WAIT ENTRY
         DFHTC TYPE=(WAIT)
         L     TIOABAR,TCTTEDA          LOAD TERM I/O AREA ADDR REG
         CLC   =C'DUMP',TIOATID
         BE    DCPDPTS                  GO TO DUMP TRANSACTION STORAGE
         CLC   =C'EOD',TIOADBA          COMP DATA FOR EOD INDICATION
         BE    DCPEXIT                  GO TO EXIT IF EQUAL
         CLC   =C'SUSPEND',TIOADBA      COMPARE FOR SUSPEND REQUEST
         BNE   DCPSRBN                  GO TO SUSPEND REQUEST BYPASS
```

```
         MVC   TWATSRL,=H'32'           MOVE TEMP STRG RECORD LENGTH
         MVC   TCATSDI(4),=C'DSDC'      MOVE TEMP STRG DATA IDENT
         MVC   TCATSDI+4(4),TCTTETI     MOVE TEMP STRG DATA IDENT
         CLC   =C'MAIN',TIOASSF
         BNE   DCPSRMB                  GO TO MAIN STRG FACILITY BYPASS
         DFHTS TYPE=PUT,                                              *
               TSDADDR=TWATSRL,                                       *
               STORFAC=MAIN
         B     DCPSRAB                  GO TO AUX STRG FACILITY BYPASS
DCPSRMB  EQU   *                        MAIN STORAGE FACILITY BYPASS
         DFHTS TYPE=PUT,                                              *
               TSDADDR=TWATSRL,                                       *
               STORFAC=AUXILIARY
DCPSRAB  EQU   *                        AUX STORAGE FACILITY BYPASS
         DFHTS TYPE=CHECK,                                            *
               NORESP=DCPSRNR
         DFHPC TYPE=ABEND,                                            *
               ABCODE=XDCS
DCPSRNR  EQU   *                        SUSPEND REQUEST NORMAL RESPONSE
         MVC   TIOATDL(DCPSRAL),DCPSRAM MOVE TERMINAL MESSAGE TO OUTPUT
         DFHTC TYPE=(WRITE)
         B     RETURN                   GO TO RETURN ENTRY
DCPSRBN  EQU   *                        SUSPEND REQUEST BYPASS ENTRY
         MVC   TCATDDI,TWATDDI          MOVE DESTINATION IDENTIFICATION
         XC    TCTTEDA,TCTTEDA          RESET TERMINAL DATA ADDRESS
         DFHTC TYPE=(READ)
         LH    14,TIOATDL               LOAD TERMINAL DATA LENGTH
         LA    14,4(0,14)               INCREMENT TERMINAL DATA LENGTH
         STH   14,TIOATDL               STORE TERMINAL DATA LENGTH
         DFHTD TYPE=PUT,                                              *
               TDADDR=TIOATDL,                                        *
               NORESP=DCPNRCN,                                        *
               IDERROR=DCPDIEN
         DFHPC TYPE=ABEND,                                            *
               ABCODE=XDCP
*****************************************************************
DCPNRCN  DS    0H                       NORMAL RESP CODE ENTRY ADDRESS
         ST    TIOABAR,TCASCSA          STORE TERM I/O AREA ADDRESS
         DFHSC TYPE=FREEMAIN
         B     DCPTEWN                  GO TO TERM EVENT WAIT ENTRY
*****************************************************************
         SPACE 4                        LISTING CONTROL CARD - SPACE 4
*****************************************************************
DCPDPTS  EQU   *                        DUMP TRANSACTION STOR ROUTINE
         DFHDC TYPE=TRANSACTION,DMPCODE=TRAN
         XC    TCTTEDA,TCTTEDA          CLEAR TERMINAL DATA AREA ADDR
         DFHTC TYPE=(READ)
         B     DCPNRCN                  RETURN TO MAINSTREAM LOGIC
*****************************************************************
         SPACE 4
*****************************************************************
DCPEXIT  EQU   *                        EXIT
         MVC   TIOATDL((4+L'DCPEODMD)),DCPEODML
         DFHTC TYPE=(WRITE)
         B     RETURN                   GO TO RETURN ENTRY
*****************************************************************
         EJECT                          LISTING CONTROL CARD - EJECT
*****************************************************************
*                          M E S S A G E   E N T R Y            *
*****************************************************************
         DC    CL32'MESSAGE ENTRY PROGRAM'
*****************************************************************
ALPMEPN  DS    0H                       MESSAGE ENTRY PROGRAM ENTRY
         MVC   TCATDDI,TIOADID          MOVE DESTINATION IDENTIFICATION
         MVC   TIOATID,TCTTETI          MOVE SOURCE IDENTIFICATION
```

```
         LH    14,TIOATDL                LOAD TERMINAL DATA LENGTH
         LA    14,4(0,14)                INCREMENT TERMINAL DATA LENGTH
         STH   14,TIOATDL                STORE TERMINAL DATA LENGTH
         DFHTD TYPE=PUT,                                              *
               TDADDR=TIOATDL,                                        *
               NORESP=MEPNRCN,                                        *
               IDERROR=MEPDIEN
         DFHPC TYPE=ABEND,                                            *
               ABCODE=XMEP
*****************************************************************
MEPNRCN  DS    0H                        NORMAL RESP CODE ENTRY ADDRESS
         MVC   TIOATDL((4+L'MEPMEAMD)),MEPMEAML
         DFHTC TYPE=(WRITE)
         B     RETURN                    GO TO RETURN ENTRY
*****************************************************************
         EJECT                           LISTING CONTROL CARD EJECT
*****************************************************************
*                        M E S S A G E   R E T R I E V A L      *
*****************************************************************
         DC    CL32'MESSAGE RETRIEVAL PROGRAM'
*****************************************************************
         SPACE 4                         LISTING CONTROL CARD - SPACE 4
*****************************************************************
ALPMRPN  DS    0H                        MESSAGE RETRIEVAL PROGRAM ENTRY
         MVC   TWAREAI,TIOARAI2          MOVE RETRIEVE ALL INDICATOR
         MVC   TWATDDI,TCTTETI           MOVE DESTINATION IDENTIFICATION
         CLC   =C'ALL',TIOARAI1          COMPARE ALL INDICATOR FOR ALL
         BNE   MRPAI1B
         MVC   TWAREAI,TIOARAI1          MOVE RETRIEVE ALL INDICATOR
         B     MRPDEBN
MRPAI1B  DS    0H                        ALL INDICATOR 1 BYPASS
         CLC   =CL4'    ',TIOADID        COMPARE DEST IDENT TO BLANKS
         BE    MRPDEBN                   GO TO DEST ID = BL IF EQUAL
         MVC   TWATDDI,TIOADID           MOVE DESTINATION IDENTIFICATION
MRPDEBN  DS    0H                        DESTINATION IDENT EQUALS BLANKS
*****************************************************************
         SPACE 4                         LISTING CONTROL CARD - SPACE 4
*****************************************************************
MRPGTDN  DS    0H                        GET TRANSIENT DATA ENTRY
         MVC   TCATDDI,TWATDDI           MOVE DESTINATION IDENTIFICATION
         DFHTD TYPE=GET,                                              *
               NORESP=MRPNRCN,                                        *
               QUEZERO=MRPQERN,                                       *
               IDERROR=MRPDIEN
         DFHPC TYPE=ABEND,                                            *
               ABCODE=XMRP
*****************************************************************
         SPACE 2                         LISTING CONTROL CARD - SPACE 2
*****************************************************************
MRPNRCN  DS    0H                        NORMAL RESP CODE ENTRY ADDRESS
         L     TDIABAR,TCATDAA           LOAD TRANS DATA AREA ADDRESS
         DFHTC TYPE=(WAIT)
         MVC   MRPMTDI+1(1),TDIAIRL+1    MOVE DATA LENGTH TO MOVE INSTR
MRPMTDI  MVC   TIOATDL(0),TDIAIRL        MOVE TRANS DATA TO TERM AREA
         LH    14,TIOATDL                LOAD TERMINAL DATA LENGTH
         SH    14,=H'4'                  SUBTRACT 4 FROM LENGTH
         STH   14,TIOATDL                STORE TERMINAL DATA LENGTH
         DFHTC TYPE=(WRITE,                                           *
               SAVE)
         CLC   =CL3'ALL',TWAREAI         COMPARE RETRIEVE ALL IND TO ALL
         BNE   RETURN                    GO TO RETURN ENTRY IF NOT EQUAL
         MVI   TWAQEMCI,X'FF'            MOVE MESSAGE CONTROL INDICATOR
         B     MRPGTDN                   GO TO GET TRANSIENT DATA ENTRY
```

```
*********************************************************************
         SPACE 4                           LISTING CONTROL CARD - SPACE 4
*********************************************************************
MRPQERN  DS    0H                          DESTINATION QUEUE EMPTY ENTRY
         CLI   TWAQEMCI,X'FF'              COMPARE MESSAGE CONTROL IND
         BE    MRPNMQMB                    GO TO NO MSG QUEUED MSG BYPASS
         MVC   TIOATDL(MRPNQML),MRPNMQM    MOVE TERMINAL MESSAGE TO OUTPUT
         B     MRPWRCS                     GO TO WRITE & RETURN TO C S
MRPNMQMB DS    0H                          NO MESSAGES QUEUED MSG BYPASS
         DFHTC TYPE=(WAIT)
         MVC   TIOATDL(MRPNMML),MRPNMMM    MOVE NO MORE MESSAGE TO T O A
*********************************************************************
MRPWRCS  DS    0H                          WRITE AND RETURN TO CONT SYS
         DFHTC TYPE=(WRITE)
         B     RETURN                      GO TO RETURN ENTRY
*********************************************************************
         EJECT                             LISTING CONTROL CARD - EJECT
*********************************************************************
* *                                                               * *
*********************************************************************
DCPDIEN  DS    0H                          DESTINATION IDENT ERROR ENTRY
         ST    TIOABAR,TCTTEDA             STORE TERM I/O AREA ADDRESS
MEPDIEN  DS    0H                          DESTINATION IDENT ERROR ENTRY
MRPDIEN  DS    0H                          DESTINATION IDENT ERROR ENTRY
         MVC   TIOATDL(MCPDEML),MCPDIEM    MOVE TERMINAL MESSAGE TO OUTPUT
         DFHTC TYPE=(WRITE)
*********************************************************************
         SPACE 4                           LISTING CONTROL CARD - SPACE 4
RETURN   DS    0H                          RETURN TO CONTROL SYSTEM
         DFHPC TYPE=RETURN
*********************************************************************
         LTORG *                           LITERAL ORIGIN AT *
*********************************************************************
         END   CICSATP                     END OF ASSEMBLY - APPL TEST PGM
```

```
*****************************************************************
   A N S      C O B O L      E X A M P L E      P R O B L E M
*****************************************************************

       DFHCOVER
       IDENTIFICATION DIVISION.
       PROGRAM-ID.
           'CICSATP'.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  MESSG1.
           02  MCPDIEM PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 60.
           02  FILL1 PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS ZERO.
           02  MESSAGE1.
             03  FILL2 PICTURE X VALUE IS ' '.
             03  FILL3 PICTURE X(8) VALUE IS ALL ' '.
             03  FILL4 PICTURE X(50) VALUE IS
                 'DESTINATION IDENTIFICATION ERROR - PLEASE RESUBMIT'.
             03  FILL5 PICTURE X VALUE IS ' '.
       01  MCPDEML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 64.
       01  DCPDCAML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 58.
       01  DCPDCAMD PICTURE X(58) VALUE IS
           ' DATA COLLECTION HAS BEEN REQUESTED AND IS ABOUT TO BEGIN '.
       01  DCPEODML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 73.
       01  DCPEODMD PICTURE X(74) VALUE IS ' THE DATA HAS BEEN RECEIVED
      -    'AND DISPATCHED TO THE DESIGNATED DESTINATION  '.
       01  MEPMEAML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 77.
       01  MEPMEAMD PICTURE X(77) VALUE IS 'YOUR MESSAGE HAS BEEN RECEIV
      -    'ED AND DISPATCHED TO THE DESIGNATED DESTINATION  '.
       01  MESSG2.
           02  MRPNMMM PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 64.
           02  FILL11 PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS ZERO.
           02  MESSAGE2.
             03  FILL21 PICTURE X VALUE IS ' '.
             03  FILL31 PICTURE X(8) VALUE IS ALL ' '.
             03  FILL41 PICTURE X(54) VALUE IS 'THERE ARE NO MORE MESSAG
      -      'ES QUEUED FOR THIS DESTINATION'.
             03  FILL51 PICTURE X VALUE IS ' '.
       01  MRPNMML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 68.
       01  MESSG3.
           02  MRPNMQM PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 59.
           02  FILL12 PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS ZERO.
           02  MESSAGE3.
             03  FILL22 PICTURE X VALUE IS ' '.
             03  FILL32 PICTURE X(8) VALUE IS ALL ' '.
             03  FILL42 PICTURE X(49) VALUE IS
                 'THERE ARE NO MESSAGES QUEUED FOR THIS DESTINATION'.
             03  FILL52 PICTURE X VALUE IS ' '.
       01  MRPNQML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 63.
       LINKAGE SECTION.
       01  DFHBLLDS COPY DFHELLDS.
           02  TCTTEAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
           02  TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
           02  TDIABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
       01  DFHCSADS COPY DFHCSADS.
       01  DFHTCADS COPY DFHTCADS.
           02  TWATDDI PICTURE X(4).
           02  TWAREAI PICTURE X(4).
           02  TWAQEMCI PICTURE S9 USAGE IS COMPUTATIONAL.
       01  DFHTCTTE COPY DFHTCTTE.
       01  DFHTIOA COPY DFHTIOA.
           02  TIOADATA.
             03  FILLER PICTURE X(80).
           02 FILLER REDEFINES TIOADATA.
```

```
      03 EODTEST PICTURE X(3).
    02  FILLER REDEFINES TIOADATA.
      03  TIOATID PICTURE X(4).
      03  FILLER PICTURE X.
      03  TIOADID.
        04  FILLER PICTURE X(4).
      03  FILLER REDEFINES TIOADID.
        04  TIOARAI1 PICTURE X(3).
      03  FILLER PICTURE X.
      03  TIOARAI2.
        04  FILLER PICTURE X(3).
      03  FILLER REDEFINES TIOARAI2.
        04  TIOAMBA PICTURE X.
01  DFHTDIA COPY DFHTDIA.
    02  TDIADBA PICTURE X(80).
PROCEDURE DIVISION.
ATPIPIN.
    MOVE CSACDTA TO TCACBAR.
    MOVE TCAFCAAA TO TCTTEAR.
    MOVE TCTTEDA TO TIOABAR.
    IF TIOATID = 'BSDC' GO TO ALPDCPN.
    IF TIOATID = 'BSME' GO TO ALPMEPN.
    IF TIOATID = 'BSMR' GO TO ALPMRPN.
  DFHPC TYPE=ABEND,                                              *
        ABCODE=XAPT
    NOTE   DATA COLLECTION PROGRAM ***.
ALPDCPN.  MOVE TIOADID TO TWATDDI.
    MOVE DCPDCAML TO TIOATDL.
    MOVE DCPDCAMD TO TIOADATA.
  DFHTC TYPE=(WRITE,READ,WAIT)
DCPTEWN.
    MOVE TCTTEDA TO TIOABAR.
    IF EODTEST = 'EOD' GO TO DCPEXIT.
    MOVE TWATDDI TO TCATDDI.
    MOVE ZEROES TO TCTTEDA.
  DFHTC TYPE=(READ,WAIT)
    ADD 4 TO TIOATDL.
  DFHTD TYPE=PUT,                                                *
        TDADDR=TIOATDL,                                          *
        NORESP=DCPNRCN,                                          *
        IDERROR=DCPDIEN
  DFHPC TYPE=ABEND,                                              *
        ABCODE=XDCP
DCPNRCN.
    MOVE TIOABAR TO TCASCSA.
  DFHSC TYPE=FREEMAIN
    GO TO DCPTEWN.
DCPEXIT.
    MOVE DCPEODML TO TIOATDL.
    ADD 4 TO TIOATDL.
    MOVE DCPEODMD TO TIOADATA.
  DFHTC TYPE=WRITE
    GO TO RETURN1.
    NOTE  MESSAGE ENTRY PROGRAM ***.
ALPMEPN.
    MOVE TIOADID TO TCATDDI.
    MOVE TCTTETI TO TIOATID.
    ADD 4 TO TIOATDL.
  DFHTD TYPE=PUT,                                                *
        TDADDR=TIOATDL,                                          *
        NORESP=MEPNRCN,                                          *
        IDERROR=MEPDIEN
  DFHPC TYPE=ABEND,                                              *
        ABCODE=XMEP
MEPNRCN.
```

```
    MOVE MEPMEAML TO TIOATDL.
    ADD 4 TO TIOATDL.
    MOVE MEPMEAMD TO TIOADATA.
  DFHTC TYPE=WRITE
    GO TO RETURN1.
    NOTE  MESSAGE RETRIEVAL PROGRAM ***.
ALPMRPN.
    MOVE TIOARAI2 TO TWAREAI.
    MOVE TCTTETI TO TWATDDI.
    IF TIOARAI1 NOT EQUAL 'ALL' GO TO MRPAI1B.
    MOVE TIOARAI11 TO TWAREAI.
    GO TO MRPDEBN.
MRPAI1B.
    IF TIOADID EQUAL ' ' GO TO MRPDEBN.
    MOVE TIOADID TO TWATDDI.
MRPDEBN.
MRPGTDN.
    MOVE TWATDDI TO TCATDDI.
  DFHTD TYPE=GET,                                                *
        NORESP=MRPNRCN,                                          *
        QUEZERO=MRPQERN,                                         *
        IDERROR=MRPDIEN
  DFHPC TYPE=ABEND,                                              *
        ABCODE=XMRP
MRPNRCN.
    MOVE TCATDAA TO TDIABAR.
    MOVE TDIAIRL TO TIOATDL.
    MOVE TDIADBA TO TIOADATA.
    SUBTRACT 4 FROM TIOATDL.
  DFHTC TYPE=(WRITE,WAIT,SAVE)
    IF TWAREAI NOT EQUAL 'ALL' GO TO RETURN1.
    MOVE 255 TO TWAQEMCI.
    GO TO MRPGTDN.
MRPQERN.
    IF TWAQEMCI EQUAL 255 GO TO MRPNMQMB.
    MOVE MRPNMQM TO TIOATDL.
    MOVE MESSAGE3 TO TIOADATA.
    GO TO MRPWRCS.
MRPNMQMB.
    MOVE MRPNMMM TO TIOATDL.
    MOVE MESSAGE2 TO TIOADATA.
MRPWRCS.
  DFHTC TYPE=WRITE
    GO TO RETURN1.
DCPDIEN.
    MOVE TIOABAR TO TCTTEDA.
MEPDIEN.
MRPDIEN.
    MOVE MCPDIEM TO TIOATDL.
    MOVE MESSAGE1 TO TIOADATA.
  DFHTC TYPE=WRITE
RETURN1.
  DFHPC TYPE=RETURN
```

```
*********************************************************************
          P L / I        E X A M P L E        P R O B L E M
*********************************************************************

 /* PL/I EXAMPLE PROBLEM */
    DFHCOVER
 CICSATP:  PROCEDURE OPTIONS (MAIN,REENTRANT);
   %INCLUDE DFHCSADS;
   %INCLUDE DFHTCADS;
       2 TWATDDI CHAR (4),
       2 TWAREAI CHAR (4),
       2 TWAQEMCI BINARY FIXED (8);
   %INCLUDE DFHTCTTE;
   %INCLUDE DFHTIOA;
       2 TIOADATA CHAR (80);
     DECLARE 1 TIOA1 BASED (TIOABAR),
               2 FILL1 CHAR (12),
               2 TIOATID CHAR (4),
               2 FILL2 CHAR (1),
               2 TIOARAI1 CHAR (3),
               2 FILL3 CHAR (2),
               2 TIOAMBA CHAR (1);
     DECLARE 1 TIOA2 BASED (TIOABAR),
               2 FILL1 CHAR (12),
               2 EODTEST CHAR (3),
               2 FILL2 CHAR (2),
               2 TIOADID CHAR (4),
               2 FILL3 CHAR (1),
               2 TIOARAI2 CHAR (3);
   %INCLUDE DFHTDIA;
       2 TDIADBA CHAR (80);
     DECLARE 1 MCPDEML BINARY FIXED (15) INITIAL (60);
     DECLARE 1 MCPDIEM CHAR(60) INITIAL ('      DESTINATION IDENTIFI
     CATION ERROR - PLEASE RESUBMIT ');
     DECLARE 1 DCPDCAML BINARY FIXED (15) INITIAL (59);
     DECLARE 1 DCPDCAMD CHAR(59) INITIAL (' DATA COLLECTION HAS BEEN RE
     QUESTED AND IS ABOUT TO BEGIN  ');
     DECLARE 1 DCPEODML BINARY FIXED (15) INITIAL (74);
     DECLARE 1 DCPEODMD CHAR (74) INITIAL (' THE DATA HAS BEEN RECEIVED
     AND DISPATCHED TO THE DESIGNATED DESTINATION  ');
     DECLARE 1 MEPMEAML BINARY FIXED (15) INITIAL (77);
     DECLARE 1 MEPEAMD CHAR(77) INITIAL (' YOUR MESSAGE HAS BEEN RECEIV
     ED AND DISPATCHED TO THE DESIGNATED DESTINATION  ');
     DECLARE 1 MRPNMML BINARY FIXED (15) INITIAL (64);
     DECLARE 1 MRPNMMM CHAR(64) INITIAL ('      THERE ARE NO MORE ME
     SSAGES QUEUED FOR THIS DESTINATION ');
     DECLARE 1 MRPNQML BINARY FIXED (15) INITIAL (59);
     DECLARE 1 MRPNMQN CHAR(59) INITIAL ('      THERE ARE NO MESSAGE
     S QUEUED FOR THIS DESTINATION ');
 ATPIPIN:  TCTTEAR = TCAFCAAA;
           TIOABAR = TCTTEDA;
           IF (TIOATID = 'PSDC') THEN GO TO ALPDCPN;
           IF (TIOATID = 'PSME') THEN GO TO ALPMEPN;
           IF (TIOATID = 'PSMR') THEN GO TO ALPMRPN;
         DFHPC TYPE=ABEND,                                          *
               ABCODE=XAPT
 /* DATA COLLECTION PROGRAM */
 ALPDCPN:  TWATDDI = TIOADID;
           TIOATDL = DCPDCAML;
           TIOADATA = DCPDCAMD;
         DFHTC TYPE= (WRITE,READ,WAIT)
 DCPTEWN:
           TIOABAR = TCTTEDA;
           IF (EODTEST = 'EOD') THEN GO TO DCPEXIT;
           TCATDDI = TWATDDI;
```

```
          UNSPEC (TCTTEDA) = 0;
        DFHTC TYPE=(READ,WAIT)
          TIOATDL = TIOATDL + 4;
        DFHTD TYPE=PUT,                                                 *
              TDADDR=TIOATDL,                                           *
              NORESP=DCPNRCN,                                           *
              IDERROR=DCPDIEN
        DFHPC TYPE=ABEND,                                               *
              ABCODE=XDCP
DCPNRCN:  TCASCSA = TIOABAR;
        DFHSC TYPE=FREEMAIN
          GO TO DCPTEWN;
  DCPEXIT: TIOATDL = DCPEODML;
          TIOADATA = DCPEODMD;
        DFHTC TYPE=WRITE
          GO TO RETURN;
/* MESSAGE ENTRY PROGRAM */
ALPMEPN:  TCATDDI = TIOADID;
          TIOATID = TCTTETI;
          TIOATDL = TIOATDL + 4;
        DFHTD TYPE=PUT,                                                 *
              TDADDR=TIOATDL,                                           *
              NORESP=MEPNRCN,                                           *
              IDERROR=MEPDIEN
        DFHPC TYPE=ABEND,                                               *
              ABCODE=XMEP
  MEPNRCN: TIOATDL = MEPMEAML; TIOADATA = MEPEAMD;
        DFHTC TYPE=WRITE
          GO TO RETURN;
  /* MESSAGE RETRIEVAL PROGRAM */
  ALPMRPN: TWAREAI = TIOARAI2; TWATDDI = TCTTETI;
          IF (TIOARAI1  ¬= 'ALL') THEN GO TO MRPAI1B;
          TWAREAI = TIOARAI1;
          GO TO MRPDEBN;
  MRPAI1B: IF (TIOADID = '    ') THEN GO TO MRPDEBN;
          TWATDDI = TIOADID;
  MRPDEBN: MRPGTDN: TCATDDI = TWATDDI;
        DFHTD TYPE=GET,                                                 *
              NORESP=MRPNRCN,                                           *
              QUEZERO=MRPQERN,                                          *
              IDERROR=MRPDIEN
        DFHPC TYPE=ABEND,                                               *
              ABCODE=XMRP
MRPNRCN:  TDIABAR = TCATDAA;
          TIOATDL = TDIAIRL - 4;
          TIOADATA = TDIADBA;
        DFHTC TYPE=(WRITE,WAIT,SAVE)
          IF (TWAREAI ¬= 'ALL ') THEN GO TO RETURN;
          TWAQEMCI = '11111111'B;
          GO TO MRPGTDN;
MRPQERN:  IF (TWAQEMCI = '11111111'B) THEN GO TO MRPNMQMB;
          TIOATDL = MRPNQML;
          TIOADATA = MRPNMQN;
          GO TO MRPWRCS;
MRPNMQMB: TIOATDL = MRPNMML; TIOADATA = MRPNMMM;
MRPWRCS:
        DFHTC TYPE=WRITE
          GO TO RETURN;
DCPDIEN:  TCTTEDA = TIOABAR;
MEPDIEN: MRPDIEN: TIOATDL = MCPDEML;
          TIOADATA = MCPDIEM;
        DFHTC TYPE=WRITE
RETURN:
          END;
```

APPENDIX B. SUMMARY OF CICS/VS STORAGE AREAS

CICS/VS areas are summarized on the charts on the following pages. The CSA and the TCTTE are in the CICS/VS nucleus; all other areas are in dynamic storage. As shown on the first chart, each area other than the VSWA consists of a control section and a user's section. The VSWA includes only a control section. Some areas are acquired by CICS/VS, some by the user (application program), and some by either CICS/VS or the application program. Assembler-language statements to obtain areas and establish addressability are shown on the first chart. Equivalent American National Standard (ANS) COBOL and PL/I statements are suggested below. For additional details, see Chapters 2 through 5 of this manual.

All CICS/VS pointers (areas containing addresses) are four bytes in length. CICS/VS uses lengths stored in two bytes; thus, the characters LL00 represent a two-byte length area followed by two bytes containing blanks.

The second chart is an expansion of the control sections that appear on the first chart. Symbolic labels that the application programmer may need to know are shown on this chart, and brief explanations of them are provided. All symbolic labels appearing on the first chart are shown in a similar manner in Appendix D. Labels on the second chart that also appear on the first chart or elsewhere in this manual are repeated in Appendix D.

The letters A through G on the first chart are keys to the following notes:

A Assembler language only.

B The TCAFCAAA may point to the address of a DCT entry or to the address of an automatic initiate descriptor (AID).

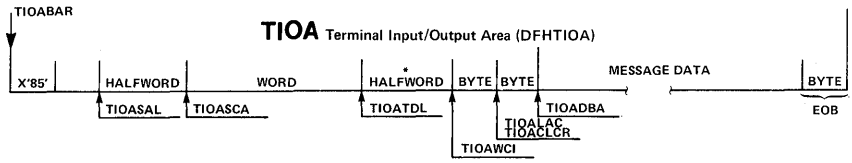
C ANS COBOL equivalent:
```
01 DFHTCTTE COPY DFHTCTTE.
01 MOVE TCAFCAAA TO TCTTEAR.
```
PL/I equivalent:
```
%INCLUDE DFHTCTTE;
```

D EOB = End of Block.

E TCAFCAA, TCATSDA, and TCATDAA: The same location (TCASCSA) within the TCA is used for these three pointers, only one of which is current at any given time.

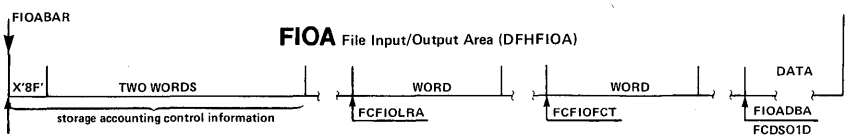
F TCASCSA may also point to an area to be released by a DFHSC TYPE=FREEMAIN macro instruction.

G After a DFHPC TYPE=LOAD macro instruction, TCAPCLA points to the beginning address of the loaded program.



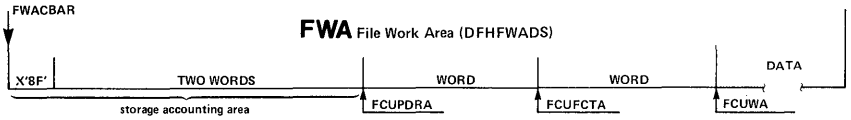
TIOABAR – TIOA Base Address Register
TIOACLCR – TIOA Control write – Line or Copy Request (same as TIOALAC)
TIOADBA – TIOA Data Begin Address
TIOALAC – TIOA Line Address Control (same as TIOACLCR)

TIOASAL – TIOA Storage Accounting – area Length
TIOASCA – TIOA Storage Chain Address
TIOATDL – TIOA Terminal – message Data Length
TIOAWC: – TIOA Write Control Indicator



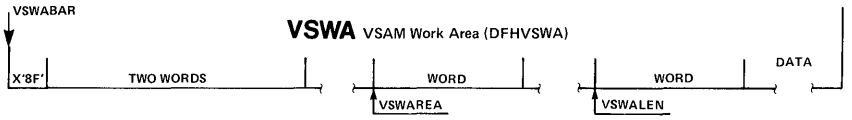
FIOABAR – File Input/Output Area Base Address Register
FCFIOxxx – File Control File Input/Output xxx
FCFIOFCT – FCFIO File Control Table – entry address

FCFIOLRA – FCFIO Logical Record Address
FIOADBA – File Input/Output Area Data Begin Address (DOS)
FCDSO1D – File Control Data – area (OS variable)

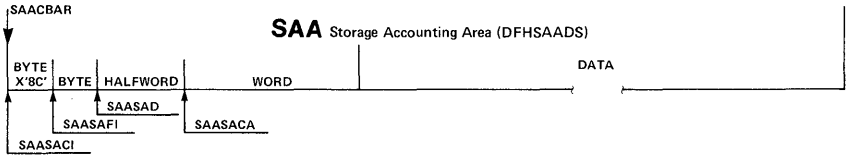


FWACBAR – File Work Area Control Base Address Register
FCUFCTA – File Control Update File Control Table Address

FCUPDRA – File Control UPDate Record Address
FCUWA – File Control Update Work Area (data begin address)

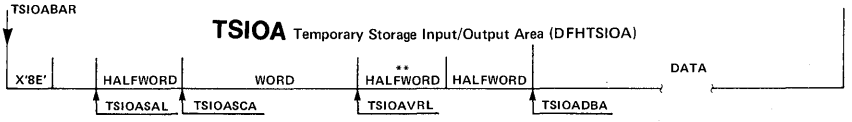


VSWABAR – VSAM Work Area Base Address Register
VSWAREA – VSAM Work Area REcord Address
VSWALEN – VSAM Work Area Record LENgth



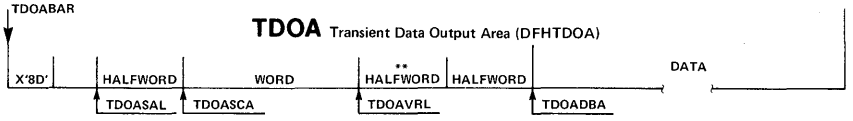
SAACBAR – SAA Control Base Address Register
SAASACA – SAA Storage Accounting Chain Address
SAASACI – SAA Storage Accounting Class Identification

SAASAFI – SAA Storage Accounting Format Identification
SAASAD – SAA Storage Accounting Displacement (length)



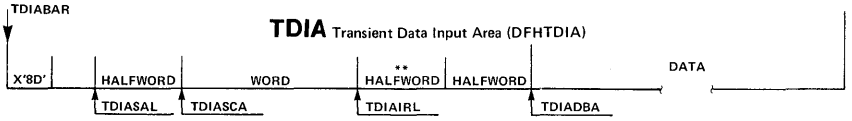
TSIOABAR – TSIOA Base Address Register
TSIOADBA – TSIOA Data Begin Address
TSIOASAL – TSIOA Storage Accounting – area Length

TSIOASCA – TSIOA Storage Chain Address
TSIOAVRL – TSIOA Variable Record Length (LLbb)**



TDOABAR – TDOA Base Address Register
TDOADBA – TDOA Data Begin Address
TDOASAL – TDOA Storage Accounting – area Length

TDOASCA – TDOA Storage Chain Address
TDOAVRL – TDOA Variable Record Length (LLbb)**



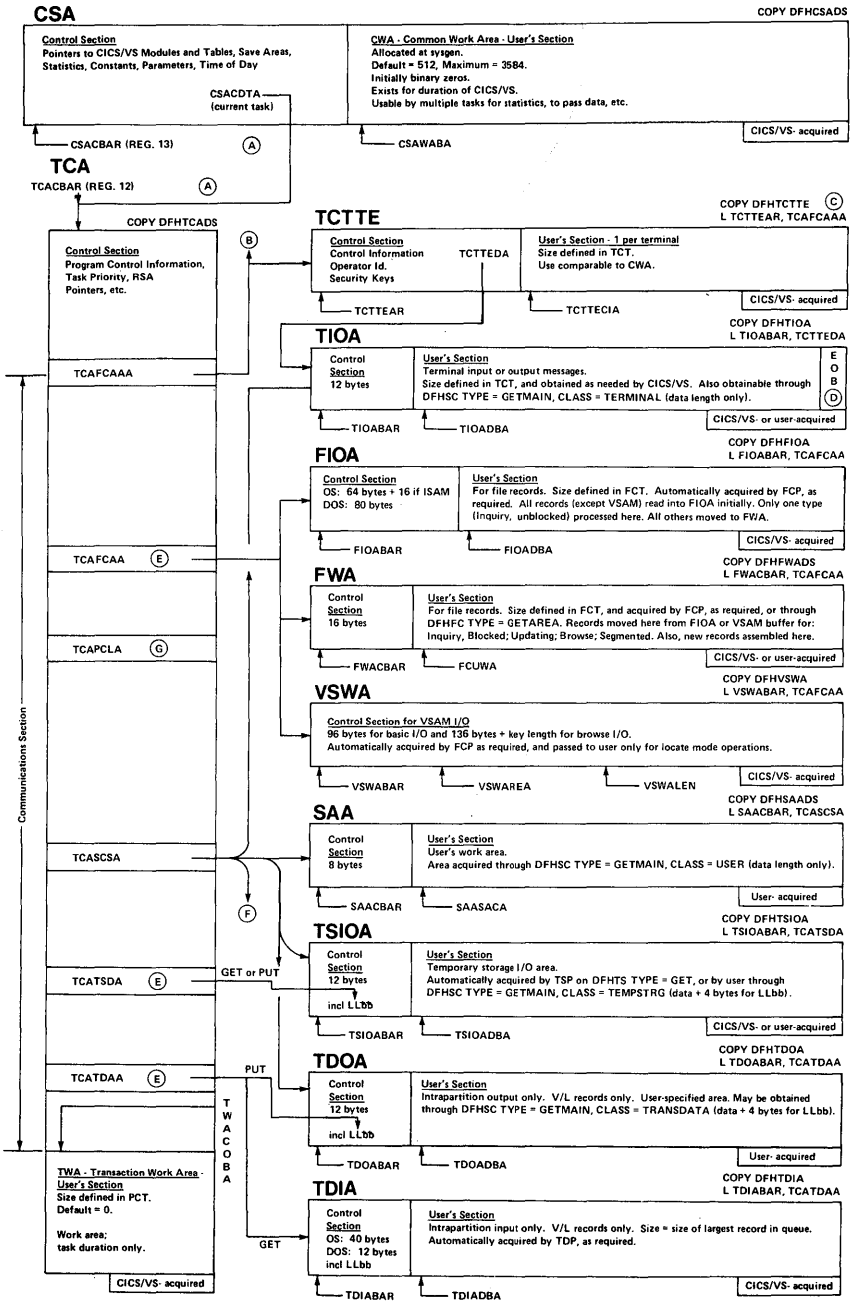
TDIABAR – TDIA Base Address Register
TDIADBA – TDIA Data Begin Address
TDIAIRL – TDIA Intrapartition Record Length (LLbb)**

TDIASAL – TDIA Storage Accounting – area Length
TDIASCA – TDIA Storage Chain Address

* Length is "Message Data" only
(does not include TIOATDL itself, or the EOB byte).

** Length includes LLbb and data.

CICS/VS AREAS – CONTROL SECTIONS



CICS/VS AREAS

APPENDIX C. EXPLANATIONS OF MNEMONICS

CICS/VS documentation contains frequent references to programs, tables, and data areas used within CICS/VS. For most of these, a shortened form of reference, or mnemonic, is established for common use. Thus, TCT stands for terminal control table, TCTTE stands for terminal control table terminal entry, and so on.

Symbolic labels are used in a similar manner in CICS/VS modules to refer to fields that contain addresses or data used by application programs. The same labels must be used in application programs referring to these fields. The labels are also designed to be mnemonics, memory aids indicating what values are represented by them. Thus, TCABMSMA refers to the task control area basic mapping support map address, TIOABAR refers to the terminal input/output area base address register, and so on.

The mnemonics referred to in the main body of this manual and apt to be encountered by application programmers writing programs to be run under CICS/VS are listed in alphabetic order in this appendix. For each, a brief explanation of the CICS/VS entity or field represented by the mnemonic is provided. Capital letters are used in the explanations to point out how the mnemonics are derived. Some of these mnemonics (and additional ones) also appear on the charts given in Appendix B. The reader should understand that this appendix is not an attempt to list all symbolic labels used within CICS/VS.

BSAM - Basic Sequential Access Method (OS/VS); in this manual, used interchangeably with SAM

BTAM - Basic Telecommunications Access Method

CCB - Command Control Block

CCC - Copy Control Character

CRDR - CICS/VS-provided Input Processor that transfers data to CICS/VS when the asynchronous transaction processing facility is used

CRLP - Card-Reader-in-Line-Printer-out terminal

CSA - Common System Area

CSACBAR - Common System Area Control Base Address Register

CSACDTA - Common System Area Currently Dispatched Task Address

CSACTODB - Common System Area Current Time of Day in Binary format

CSAJYDP - Common System Area Julian Year/Day in Packed decimal format (four bytes)

CSAOBAR - Common System Area Optional Features List Base Address Register

CSAOPFLA - Common System Area Optional Features List Address

CSATODP - Common System Area Time of Day in Packed decimal format (four bytes)

CSATRTBA - Common System Area TRace TaBle Address

CSAWABA - Common System Area Work Area Beginning Address

CSML - Control System Message Log

CSMT - Control System Master Terminal

CSSL - Control System System Log

CSTL - Control System Terminal Log

CWA - Common Work Area

CWTR - CICS/VS-provided Output Processor that transfers data from CICS/VS when the Asynchronous Transaction Processing facility is used

DCT - Destination Control Table

ECB - Event Control Block

FCA - Facility Control Area

FCFIOBEX - File Control File Input/Output BDAM Error Code (four bytes)

FCFIOEX - File Control File Input/Output ISAM Error Code (four bytes)

FCT - file control table

FCUWA - File Control Update Work Area (data begin address)

FIOA - File Input/Output Area

FIOABAR - File Input/Output Area Base Address Register

FWA - file work area

FWACBAR - file work area Control Base Address Register

ISAM - Indexed Sequential Access Method

JCA - Journal Control Area

JCAADATA - Journal Control Area Address of DATA to be written to journal data set

JCAAPRFX - Journal Control Area Address of user-PReFiX data

JCABAR - Journal Control Area Base Address Register

JCAECN - Journal Control Area Event Control Number (four bytes)

JCAJCRC - Journal Control Area Journal Control Response Code (one byte)

JCAJFID - Journal Control Area Journal File IDentification (one byte)

JCAJRTID - Journal Control Area Journal Record Type IDentification (two bytes)

JCALDATA - Journal Control Area Length of DATA to be written to journal data set (two bytes)

JCALPRFX - Journal Control Area Length of user PReFiX (two bytes)

JCARCIDE - Journal Control Area Response Code IDentification Error (Assembler only; cne byte, X'01')

JCARCIOE - Journal Control Area Response Code Input/Output Error (Assembler cnly; one byte, X'07')

JCARCIRE - Journal Control Area Response Code Invalid Request Error (Assembler only; one byte, X'02')

JCARCLE - Journal Control Area Response Code Length Error (Assembler only; one byte, X'06')

JCARCNOE - Journal Control Area Response Code data set Not Open Error (Assembler only; one byte, X'05')

JCARCNR - Journal Control Area Response Code Normal Response (Assembler only; one byte, X'00')

JCT - Journal Control Table

PAM - Page Allocation Map

PCB - Program Control Block

PCT - Program Control Table

PPT - Processing Program Table

PSB - Program Specification Block

RBA - Relative Byte Address

RSA - Register Save Area

SAA - Storage Accounting Area

SAACBAR - Storage Accounting Area Control Base Address Register

SAASACA - Storage Accounting Area Storage Acccunting Chain Address

SAM - Sequential Access Method (DOS/VS); in this manual, used interchangeably with BSAM

SRT - System Recovery Table

SSA - Segment Search Argument

TCA - Task Control Area

TCABFTR - Task Control Area Built-in Function Type of Request (one byte)

TCABITF - Task Control Area BIT Manipulation address of one-byte bit Field to be operated on

TCABITR - Task Control Area BIT Manipulation Result of BITEST operation (one byte)

TCABITV - Task Control Area BIT Manipulation address of bit pattern (mask) to be applied to a specified byte

TCABMSCP - Task Control Area basic mapping support Cursor Position (two bytes)

TCABMSMA - Task Control Area basic mapping support Map Address

TCABMSMN - Task Control Area basic mapping support Map Name (eight bytes)

TCACBAR - Task Control Area Control Base Address Register

TCACCCA - Task Control Area Common Control Communication Area

TCACKFD - Task Control Area Field Verify address of FieLD to be ChecKed

TCACKLN - Task Control Area Field Verify LeNgth of field to be ChecKed (two bytes)

TCADCDC - Task Control Area Dump Control Dump Code (four bytes)

TCADCNB - Task Control Area Dump Control Number of Bytes in area to be dumped (two bytes)

TCADCSA - Task Control Area Dump Control Storage Address of area to be dumped

TCADCTR - Task Control Area Dump Control Type of Request (Assembler or PL/I; two bytes)

TCADLECB - Task Control Area DL/I Event Control Block

TCADLFUN - Task Control Area DL/I FUNction (four bytes)

TCADLIO - Task Control Area DL/I Input/Output area address

TCADLPCB - Task Control Area DL/I Program Control Block address

TCADLPSB - Task Control Area DL/I program specification block name (eight bytes)

TCADLSSA - Task Control Area DL/I address of segment search argument list

TCAFCAA - Task Control Area File Control Area Address

TCAFCAAA - Task Control Area Facility Control Area Associated Address

TCAFCAI - Task Control Area File Control indirect Access data set Identification (eight bytes)

TCAFCDI - Task Control Area File Control Data set Identification (eight bytes)

TCAFCNRD - Task Control Area File Control Number of Records Deleted (two bytes binary)

TCAFCRC - Task Control Area File Control Response Code (ANS COBOL)

TCAFCRI - Task Control Area File Control record identification (eight bytes)

TCAFCSI - Task Control Area File Control Segment Identification (eight bytes)

TCAFCTR - Task Control Area File Control Type of Request/Response (Assembler or PL/I; one byte)

TCAFCURL - Task Control Area File Control Undefined Record Length (two bytes)

TCAFLD - Task Control Area Field Edit address of FieLD to be edited

TCAFLN - Task Control Area Field Edit LeNgth of Field to be edited (two bytes)

TCAICDA - Task Control Area Interval Control Data Area

TCAICQID - Task Control Area Interval Control reQuest IDentification (eight bytes)

TCAICQPX - Task Control Area Interval Control reQuest Prefix (two bytes)

TCAICRC - Task Control Area Interval Control Response Code (ANS COBOL)

TCAICRT - Task Control Area Interval Control Request Time (four bytes)

TCAICTEC - Task Control Area Interval Control Timer Event Control area address

TCAICTI - Task Control Area Interval Control Transaction Identification (four bytes)

TCAICTID - Task Control Area Interval Control Terminal IDentification (four bytes)

TCAICTR - Task Control Area Interval Control Type of Request/Response (Assembler or PL/I; one byte)

TCAINA1 - Task Control Area INput Formatting Address of list of offsets for the internal fixed-format TIOA

TCAINA2 - Task Control Area INput Formatting Address of list of field names that may appear in input stream

TCAINH1 - Task Control Area INput Formatting length of the TIOA to be acquired for the internal fixed-format representation of data (Halfword field)

TCAINRC - Task Control Area INput Formatting Response Code (one byte)

TCAJCAAD - Task Control Area Journal Control Area ADdress

TCAKCFA - Task Control Area Task Control (KCP) Facility control area Address

TCAKCTA - Task Control Area Task Control (KCP) TCA Address

TCAKCTI - Task Control Area Task Control (KCP) Transaction Identification (four bytes)

TCAKCTTA - Task Control Area Task Control (KCP) (three bytes; only two low-order bytes are used for transaction identification)

TCAMSFMP - Task Control Area Mapping Support Function Management Parameter

TCAMSHDR - Task Control Area Mapping Support HeaDeR address (four bytes)

TCAMSIOA - Task Control Area Mapping Support Input/Output Area Address

TCAMSJ - Task Control Area Mapping Support Justification (one byte)

TCAMSMSA - Task Control Area Mapping Support Map Set Address

TCAMSMSN - Task Control Area Mapping Support Map Set Name (eight bytes)

TCAMSOC - Task Control Area Mapping Support Operator Class (three bytes)

TCAMSPGN - Task Control Area Mapping Support PaGe Number (current page; two bytes binary)

TCAMSRC1-TCAMSRC3 - Task Control Area Mapping Support Response Code (one byte each)

TCAMSRID - Task Control Area Mapping Support Request IDentification

TCAMSRI1 - Task Control Area Mapping Support Return Information (one byte)

TCAMSRLA - Task Control Area Mapping Support Routing List Address, or Returned page List Address

TCAMSRTI - Task Control Area Mapping Support Routing Time or Time interval Indicator (four bytes packed decimal)

TCAMSTA - Task Control Area Mapping Support Title Address

TCAMSTI - Task Control Area Mapping Support error Terminal Identification (four bytes)

TCAMSTRL - Task Control Area Mapping Support TRaiLer address (four bytes)

TCAMSTR1-TCAMSTR7 - Task Control Area Mapping Support Type Request (one byte each)

TCANAME - Task Control Area Phonetic Conversion 16-byte field containing data (NAME) to be phonetically encoded

TCANXTID - Task Control Area NeXt Transaction IDentification (four bytes)

TCAPCAC - Task Control Area Program Control ABEND Code (four bytes)

TCAPCERA - Task Control Area Program Control Exit Routine Address

TCAPCPI - Task Control Area Program Control Program Identification (eight bytes)

TCAPCTA - Task Control Area Program Control program Processing Program Table Address of currently executing program

TCAPHNR - Task Control Area PHoNetic Conversion error Response indicator (contains X'54' if invalid name was encountered; one byte)

TCAPHON - Task Control Area PHONetic Conversion 4-byte returned value

TCASCIB - Task Control Area Storage Control Initialization Byte

TCASCNB - Task Control Area Storage Control Number of Bytes of storage requested (two bytes)

TCASCSA - Task Control Area Storage Control Storage Address of area acquired or to be freed

TCATCDC - Task Control Area Task Control Dispatcher Control indicator (one byte)

TCATCDP - Task Control Area Task Control Dispatching Priority (one byte)

TCATCEA - Task Control Area Task Control Event control area Address (ECB or CCB or list)

TCATCQA - Task Control Area Task Control enQueued resource length (high-order byte) and Address (three low-order bytes)

TCATDAA - Task Control Area Transient Data Area Address

TCATDDI - Task Control Area Transient Data Destination Identification (four bytes)

TCATDRC - Task Control Area Transient Data Response Code (ANS COBOL)

TCATDTR - Task Control Area Transient Data Type of Request/Response (Assembler or PL/I; one byte)

TCATSAF - Task Control Area Table Search length of Field in Argument table entry to be compared with search argument (one byte)

TCATSA1 - Task Control Area Table Search Address of search argument

TCATSA2 - Task Control Area Table Search Address of first entry in argument table

TCATSA3 - Task Control Area Table Search Address of first function table entry

TCATSA4 - Task Control Area Table Search Address of field in first entry in argument table to be compared with search argument

TCATSA5 - Task Control Area Table Search Address of (1) function field within first function table entry, on input, or (2) function field within function table entry which contained value retrieved, on output

TCATSDA - Task Control Area Temporary Storage Data Address

TCATSDI - Task Control Area Temporary Storage Data Identification (eight bytes)

TCATSFF - Task Control Area Table Search length of Field in Function table entry to be retrieved (one byte)

TCATSH1 - Task Control Area Table Search maximum number of entries to be searched (halfword)

TCATSH2 - Task Control Area Table Search length of each argument table entry (halfword)

TCATSH3 - Task Control Area Table Search length of each function table entry (halfword)

TCATSH4 - Task Control Area Table Search index value (relative to 1) identifying the matching argument table entry returned to application program; if zero, no matching entry was found (halfword)

TCATSRC - Task Control Area Temporary Storage Response Code (ANS COBOL)

TCATSRN - Task Control Area Temporary Storage Record Number

TCATSRPC - Task Control Area Table Search ResPonse Code

TCATSTR - Task Control Area Temporary Storage Type of Request/Response (Assembler or PL/I; one byte)

TCAWGAA - Task Control Area WeiGhted Retrieval VSWA pointer

TCAWGCNT - Task Control Area Weighted Retrieval Count of maximum number of records to be made available to application program; NRECDS parameter (halfword field)

TCAWGH1 - Task Control Area WeiGhted Retrieval highest percentage of acceptability for a weighted retrieval function (halfword)

TCAWGH2 - Task Control Area WeiGhted Retrieval lowest percentage of acceptability for a weighted retrieval function (halfword)

TCAWGH3 - Task Control Area WeiGhted Retrieval percentage of acceptability of this record saved as the result of a weighted retrieval operation (Halfword field)

TCAWGH4 - Task Control Area WeiGhted Retrieval number of records left to be presented to user (Halfword field)

TCAWGH5 - Task Control Area WeiGhted Retrieval number of records dropped to remain within user-specified maximum (NRECDS) (Halfword field)

TCAWPA1 - Task Control Area Weighted Retrieval Address of search argument

TCAWPA3 - Task Control Area Weighted Retrieval Address of area containing record to be examined

TCAWPA4 - Task Control Area Weighted Retrieval Address of field within area containing record to be examined

TCAWPB1 - Task Control Area Weighted Retrieval character indicating format of search argument (one byte)

TCAWPH1 - Task Control Area Weighted Retrieval length of search argument (halfword)

TCAWPH2 - Task Control Area Weighted Retrieval match value (halfword)

TCAWPH3 - Task Control Area Weighted Retrieval no match value (halfword)

TCAWPH4 - Task Control Area Weighted Retrieval upper limit of comparison range (halfword)

TCAWPH5 - Task Control Area Weighted Retrieval lower limit of comparison range (halfword)

TCAWPNL - Task Control Area Weighted Retrieval NulL character (one byte)

TCAWPTR - Task Control Area Weighted Retrieval Type of Range (one byte)

TCAWRAA - Task Control Area Weighted Retrieval VSWA pointer

TCAWTDI - Task Control Area Weighted ReTrieval Data Identification (eight bytes)

TCAWTH1 - Task Control Area Weighted ReTrieval maximum number of records to be retrieved (halfword)

TCAWTH2 - Task Control Area Weighted ReTrieval relative number of record with same partial key to be examined first (halfword)

TCAWTH3 - Task Control Area Weighted ReTrieval maximum percentage of acceptability for retrieved records (halfword)

TCAWTH4 - Task Control Area Weighted ReTrieval minimum percentage of acceptability for retrieved records (halfword)

TCAWTRC - Task Control Area Weighted Retrieval Response Code (one byte)

TCAWTRI - Task Control Area Weighted ReTrieval address of partial key of Record at which retrieval is to begin (fullword)

TCT - Terminal Control Table

TCTLE - Terminal Control Table Line Entry

TCTTE - Terminal Control Table Terminal Entry

TCTTEAID - Terminal Control Table Terminal Entry Attention IDentifier (used with the 3270 Information Display System, particularly, under CICS/VS basic mapping support; one byte)

TCTTEAR - Terminal Control Table Terminal Entry Address Register

TCTTECIA - Terminal Control Table Terminal Entry Control Information Area pointer

TCTTEDA - Terminal Control Table Terminal Entry Data Address

TCTTELPL - Terminal Control Table Terminal Entry Line Printer Length (two bytes)

TCTTEOS - Terminal Control Table Terminal Entry External Operation Status (for example, READ,WRITE,WAIT request being serviced by TCP; one byte)

TCTTEPCF - Terminal Control Table Terminal Entry Passbook Control Field (2980 General Banking Terminal System; one byte)

TCTTEPCR - Terminal Control Table Terminal Entry Passbook Control Read indicator (2980 General Banking Terminal System; one byte)

TCTTEPCW - Terminal Control Table Terminal Entry Passbook Control Write indicator (2980 General Banking Terminal System; one byte)

TCTTESID - Terminal Control Table Terminal Entry Station IDentification (2980 General Banking Terminal System; one byte)

TCTTETAB - Terminal Control Table Terminal Entry TABs needed to position print element (2980 General Banking Terminal System; one byte)

TCTTETID - Terminal Control Table Terminal Entry Teller IDentification (2980 General Banking Terminal; one byte)

TCTTETM - Terminal Control Table Terminal Entry Terminal Model (one byte)

TCTTETT - Terminal Control Table Terminal Entry Terminal Entry Terminal Type (used by CICS/VS basic mapping support; one byte)

TDIA - Transient Data Input Area

TDIABAR - Transient Data Input Area Base Address Register

TDIADBA - Transient Data Input Area Data Begin Address

TDIAIRL - Transient Data Input Area Intrapartition Record Length (two bytes)

TDOA - Transient Data Output Area

TDOABAR - Transient Data Output Area Base Address Register

TDOADBA - Transient Data Output Area Data Begin Address

TDOAVRL - Transient Data Output Area Variable Record Length (two bytes)

TIOA - Terminal Input/Output Area

TIOABAR - Terminal Input/Output Area Base Address Register

TIOACLCR - Terminal Input/Output Area ControL CharacteR (same as TIOALAC; one byte)

TIOADBA - Terminal Input/Output Area Data Begin Address

TIOALAC - terminal input/output area Line Address Control (same as TIOACLCR; one byte)

TIOATDL - Terminal Input/Output Area Transmission Data Length (two bytes)

TSIOA - Temporary Storage Input/Output Area

TSIOABAR - Temporary Storage Input/Output Area Base Address Register

TSIOADBA - Temporary Storage Input/Output Area Data Begin Address

TSIOAVRL - Temporary Storage Input/Output Area Variable Record Length (two bytes)

TWA - Transaction Work Area

TWACOBA Transaction Work Area Common Origin Beginning Address

TWANXREC - Transaction Work Area address of NeXt RECord (CWTR exit routine)

TWAREC - Transaction Work Area address of RECord to be processed (CRDR or CWTR exit routine)

TWAWA - Transaction Work Area Work Area (CRDR or CWTR exit routine)

TWAXTRTN - Transaction Work Area exit ReTurN (CRDR or CWTR exit routine)

VSAM - Virtual Storage Access Method

VSWA - VSAM Work Area

VSWABAR - VSAM Work Area Base Address Register

VSWALEN - VSAM Work Area record LENgth (four bytes)

VSWAREA - VSAM Work Area REcord Address

VTAM - Virtual Telecommunication's Access Method

WCC - Write Control Character

APPENDIX D. CICS/VS MACRO INSTRUCTIONS

The general formats of CICS/VS macro instructions as presented in this manual are summarized in this appendix. The formats are shown in the same order that they appear in preceding sections of the manual.

TERMINAL SERVICES

INPUT OPERATIONS

VTAM-Supported Terminals

	DFHTC	TYPE=(READ[,WAIT][,SAVE]) [,IOTYPE={IMMED / DELAY}] [,EODS=symbolic address]

System/7

	DFHTC	TYPE=(READ[,WAIT][,SAVE][,{TRANSPARENT / PSEUDOBIN}])

2260 Display Station

	DFHTC	TYPE={READ / READL}[,WAIT][,SAVE])

2741 Communication Terminal

	DFHTC	TYPE=(READ[,WAIT]) ,RDATT=symbclic address

3270 Information Display System

	DFHTC	TYPE={READ / READL / READB}[,WAIT][,SAVE][,TEXT])

3735 Programmable Buffered Terminal

	DFHTC	TYPE=(READ[,WAIT][,SAVE]) [,EOF=symbolic address]

3740 Data Entry System

	DFHTC	TYPE=(READ[,WAIT][,SAVE]) [,ENDFILE=symbolic address] [,ENDINPT=symbolic address]

All Other CICS/VS-Supported Terminals

	DFHTC	TYPE=(READ[,WAIT][,SAVE])

OUTPUT OPERATIONS

TCAM-Supported Terminals

	DFHTC	TYPE=(WRITE[,other optional parameters]) [,DEST={symbolic name / YES}] [,terminal dependent operands]

VTAM-Supported Terminals

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE][,LAST]) [,IOTYPE={IMMED / DELAY}] [,LDC={mnemonic / YES}] [,FMH={NO / YES}]

System/3, System/370, 2770, 2780, or 3780

	DFHTC	TYPE=(WRITE[,WAIT][,SAVE][,TRANSPARENT])

System/7

```
|  | DFHTC | TYPE=(WRITE[,WAIT][,SAVE][,{TRANSPARENT}])
|  |       |                            {PSEUDOBIN  }
```

2260 Display Station

```
|  | DFHTC | TYPE=({WRITE }[,WAIT][,SAVE][,ERASE])
|  |       |       {WRITEL}
|  |       |  [,LINEADR={number}]
|  |       |            {YES   }
```

2741 Communication Terminal

```
|  | DFHTC | TYPE=(WRITE[,WAIT][,SAVE])
|  |       | ,WRBRK=symbolic address CICS/OS/VS Only
```

2980 General Banking Terminal

```
|  | DFHTC | TYPE={CBUFF }
|  |       |      {PASSBK}
```

3270 Information Display System

```
|  | DFHTC | TYPE=({WRITE  }[,WAIT][,SAVE][,ERASE])
|  |       |       {WRITEL }
|  |       |       {COPY   }
|  |       |       {PRINT  }
|  |       |       {ERASEAUP}
|  |       |  [,LINEADR={number}]
|  |       |            {YES   }
|  |       |  [,CTLCHAR={hexadecimal number}]
|  |       |            {YES               }
```

3735 Programable Buffered Terminal

```
|  | DFHTC | TYPE=(WRITE[,WAIT][,SAVE][,NOTRANSLATE])
```

3740 Data Entry System

```
|      | DFHTC | TYPE=([WRITE][,WAIT][,SAVE][,ENDFILE][,ENDOUTPUT]
|      |       |       [,TRANSPARENT])
```

All Other CICS/VS-Supported Terminals

```
|      | DFHTC | TYPE=(WRITE[,WAIT][,SAVE]
```

MISCELLANEOUS OPERATIONS

Line Control

```
|      | DFHTC | TYPE={RESET     } binary synchronous lines
|      |       |      {DISCONNECT} switched lines
```

3650 Program Request

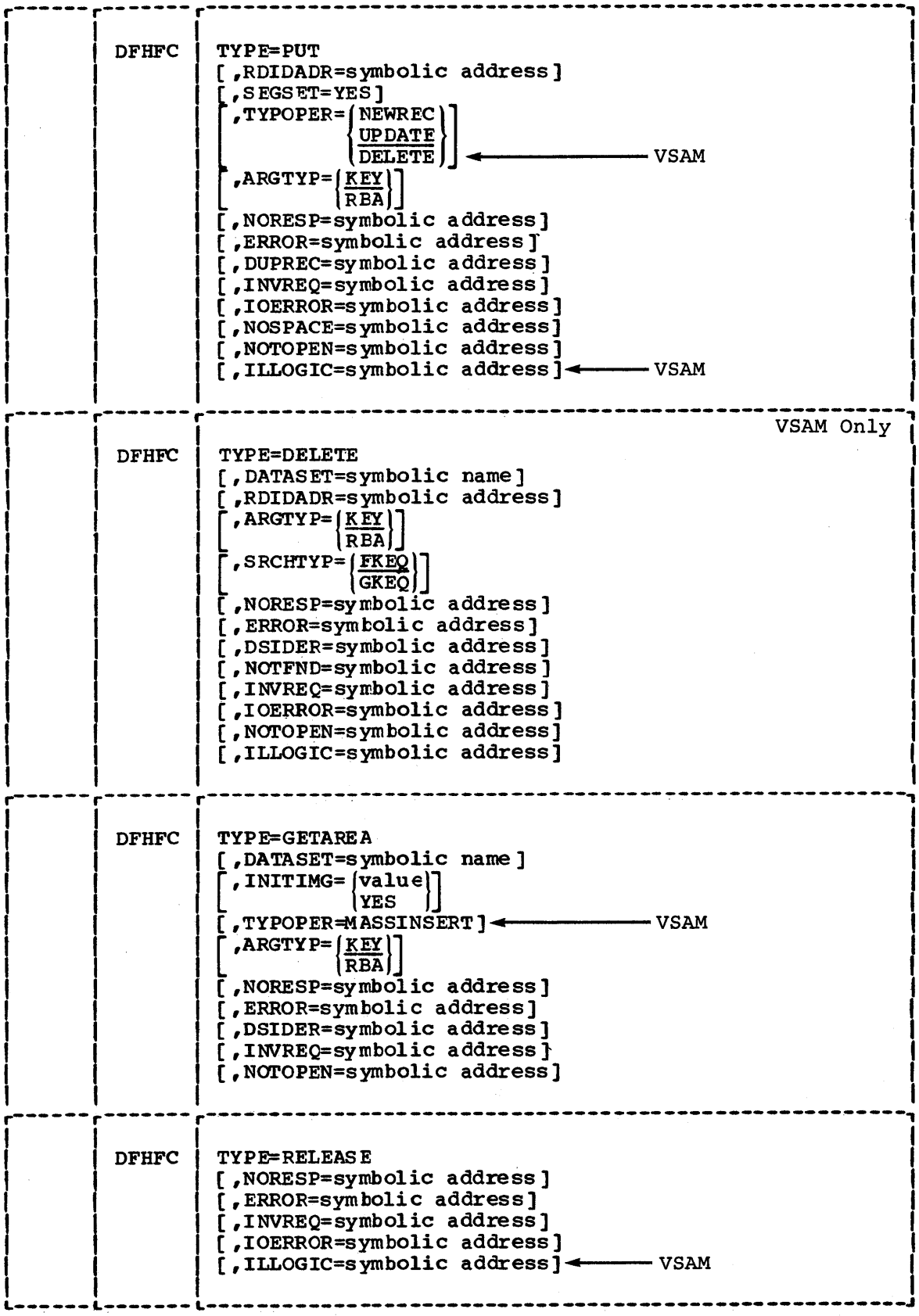
```
|      |       |
|  For 3650:   |
|      |       |
|      | DFHTC | TYPE=PROGRAM
|      |       | ,PRGNAME=name
|      |       |[,VALID=address]
|      |       |[,NONVAL=address]
|      |       |[,CONNECT={ACTIVATE}]
|      |       |[         {CONVERSE}]
|      |       |[,NORESP=address]
|      |       |
```

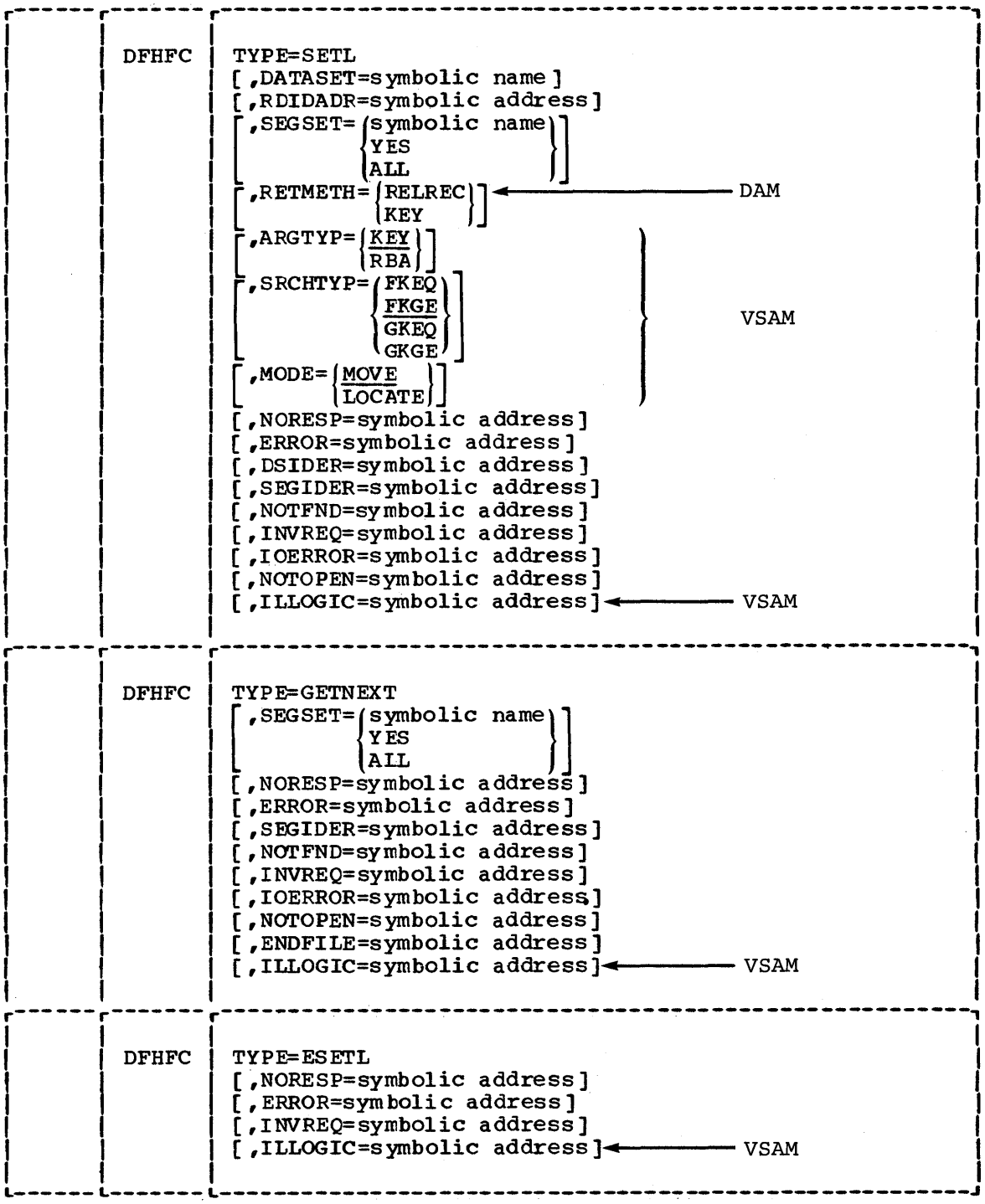
EODS for 3650 Interpreter Logical Units

```
|      |       |
|      | DFHTC | TYPE=EODS
|      |       |
```

FILE SERVICES

```
        DFHFC   TYPE=GET
                [,DATASET=symbolic name]
                [,RDIDADR=symbolic address]
                [,SEGSET={symbolic name}]
                         {YES          }
                         {ALL          }
                [,INDEX={symbolic name}]
                        {YES          }
                [,TYPOPER=UPDATE]
                [,RETMETH={RELREC}]                    DAM
                          {KEY   }
                [,ARGTYP={KEY}]
                         {RBA}
                [,SRCHTYP={FKEQ}]
                          {FKGE}
                          {GKEQ}                       VSAM
                          {GKGE}
                [,MODE={MOVE  }]
                       {LOCATE}
                [,NORESP=symbolic address]
                [,ERROR=symbolic address]
                [,DSIDER=symbolic address]
                [,SEGIDER=symbolic address]
                [,NOTFND=symbolic address]
                [,INVREQ=symbolic address]
                [,IOERROR=symbolic address]
                [,DUPDS=symbolic address]
                [,NOTOPEN=symbolic address]            VSAM
                [,ILLOGIC=symbolic address]
```


DFHFC
TYPE=PUT
[,RDIDADR=symbolic address]
[,SEGSET=YES]
[,TYPOPER={NEWREC|UPDATE|DELETE}] ← VSAM
[,ARGTYP={KEY|RBA}]
[,NORESP=symbolic address]
[,ERROR=symbolic address]
[,DUPREC=symbolic address]
[,INVREQ=symbolic address]
[,IOERROR=symbolic address]
[,NOSPACE=symbolic address]
[,NOTOPEN=symbolic address]
[,ILLOGIC=symbolic address] ← VSAM
VSAM Only
DFHFC
TYPE=DELETE
[,DATASET=symbolic name]
[,RDIDADR=symbolic address]
[,ARGTYP={KEY|RBA}]
[,SRCHTYP={FKEQ|GKEQ}]
[,NORESP=symbolic address]
[,ERROR=symbolic address]
[,DSIDER=symbolic address]
[,NOTFND=symbolic address]
[,INVREQ=symbolic address]
[,IOERROR=symbolic address]
[,NOTOPEN=symbolic address]
[,ILLOGIC=symbolic address]
DFHFC
TYPE=GETAREA
[,DATASET=symbolic name]
[,INITIMG={value|YES}]
[,TYPOPER=MASSINSERT] ← VSAM
[,ARGTYP={KEY|RBA}]
[,NORESP=symbolic address]
[,ERROR=symbolic address]
[,DSIDER=symbolic address]
[,INVREQ=symbolic address]
[,NOTOPEN=symbolic address]
DFHFC
TYPE=RELEASE
[,NORESP=symbolic address]
[,ERROR=symbolic address]
[,INVREQ=symbolic address]
[,IOERROR=symbolic address]
[,ILLOGIC=symbolic address] ← VSAM


DFHFC
TYPE=SETL
[,DATASET=symbolic name]
[,RDIDADR=symbolic address]
[,SEGSET={symbolic name / YES / ALL}]
[,RETMETH={RELREC / KEY}]
DAM
[,ARGTYP={KEY / RBA}]
[,SRCHTYP={FKEQ / FKGE / GKEQ / GKGE}]
[,MODE={MOVE / LOCATE}]
VSAM
[,NORESP=symbolic address]
[,ERROR=symbolic address]
[,DSIDER=symbolic address]
[,SEGIDER=symbolic address]
[,NOTFND=symbolic address]
[,INVREQ=symbolic address]
[,IOERROR=symbolic address]
[,NOTOPEN=symbolic address]
[,ILLOGIC=symbolic address]
VSAM
DFHFC
TYPE=GETNEXT
[,SEGSET={symbolic name / YES / ALL}]
[,NORESP=symbolic address]
[,ERROR=symbolic address]
[,SEGIDER=symbolic address]
[,NOTFND=symbolic address]
[,INVREQ=symbolic address]
[,IOERROR=symbolic address]
[,NOTOPEN=symbolic address]
[,ENDFILE=symbolic address]
[,ILLOGIC=symbolic address]
VSAM
DFHFC
TYPE=ESETL
[,NORESP=symbolic address]
[,ERROR=symbolic address]
[,INVREQ=symbolic address]
[,ILLOGIC=symbolic address]
VSAM

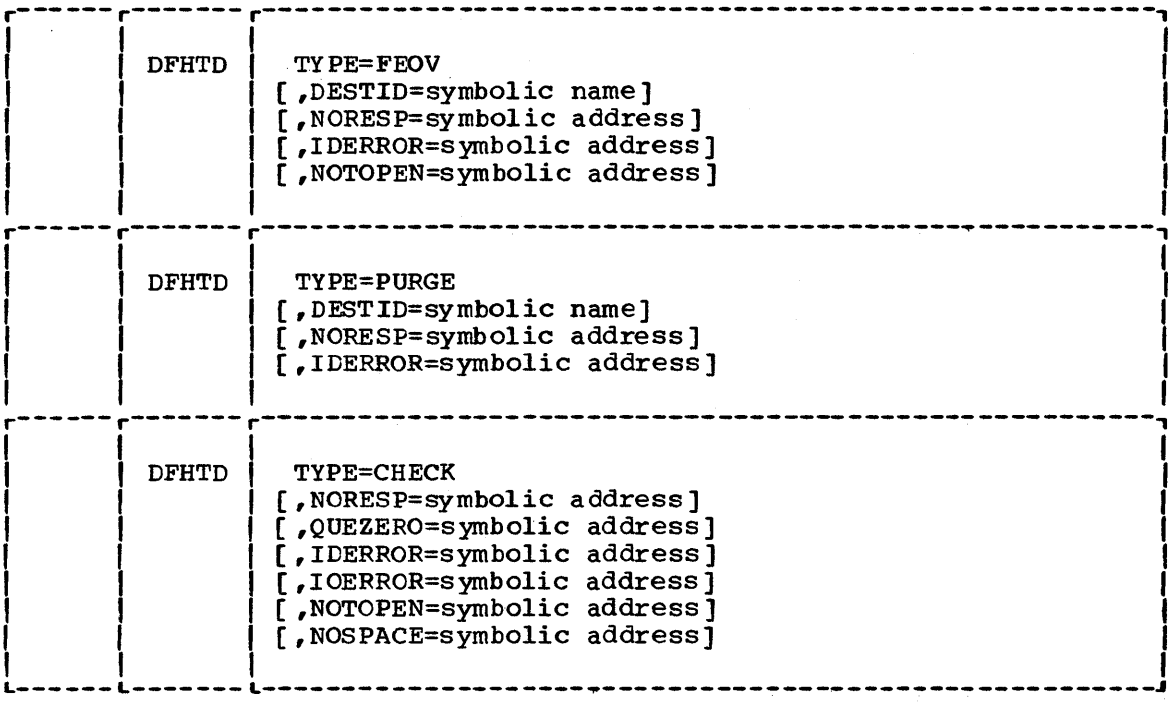
```
DFHFC   TYPE=RESETL
        [,SEGSET={symbolic name}]
                 {YES          }
                 {ALL          }
        [,ARGTYP={KEY}]                    }
                 {RBA}]                    }
        [,SRCHTYP={FKEQ}]                  }  VSAM
                  {FKGE}                   }
                  {GKEQ}                   }
                  {GKGE}]                  }
        [,NORESP=symbolic address]
        [,ERROR=symbolic address]
        [,SEGIDER=symbolic address]
        [,NOTFND=symbolic address]
        [,INVREQ=symbolic address]
        [,IOERROR=symbolic address]
        [,NOTOPEN=symbolic address]
        [,ILLOGIC=symbolic address]  <-- VSAM
```

```
DFHFC   TYPE=CHECK
        [,NORESP=symbolic address]
        [,ERROR=symbolic address]
        [,DSIDER=symbolic address]
        [,SEGIDER=symbolic address]
        [,NOTFND=symbolic address]
        [,DUPREC=symbolic address]
        [,INVREQ=symbolic address]
        [,IOERROR=symbolic address]
        [,DUPDS=symbolic address]
        [,NOSPACE=symbolic address]
        [,NOTOPEN=symbolic address]
        [,ENDFILE=symbolic address]
        [,ILLOGIC=symbolic address]  <-- VSAM
```

TRANSIENT DATA SERVICES

```
DFHTD   TYPE=PUT
        [,DESTID=symbolic name]
        [,TDADDR=symbolic address]
        [,NORESP=symbolic address]
        [,IDERROR=symbolic address]
        [,IOERROR=symbolic address]
        [,NOTOPEN=symbolic address]
        [,NOSPACE=symbolic address]
```

```
DFHTD   TYPE=GET
        [,DESTID=symbolic address]
        [,QUEBUSY=symbolic address]  <-- VSAM
        [,NORESP=symbolic address]
        [,QUEZERO=symbolic address]
        [,IDERROR=symbolic address]
        [,IOERROR=symbolic address]
        [,NOTOPEN=symbolic address]
```


DFHTD
TYPE=FEOV
[,DESTID=symbolic name]
[,NORESP=symbolic address]
[,IDERROR=symbolic address]
[,NOTOPEN=symbolic address]
DFHTD
TYPE=PURGE
[,DESTID=symbolic name]
[,NORESP=symbolic address]
[,IDERROR=symbolic address]
DFHTD
TYPE=CHECK
[,NORESP=symbolic address]
[,QUEZERO=symbolic address]
[,IDERROR=symbolic address]
[,IOERROR=symbolic address]
[,NOTOPEN=symbolic address]
[,NOSPACE=symbolic address]

TEMPORARY STORAGE SERVICES

```
        DFHTS   TYPE=PUT
                [,TYPOPER=REPLACE]
                [,DATAID=name]
                [,TSDADDR={symbolic address}]
                          {YES              }
                [,STORFAC={AUXILIARY}]
                          {MAIN      }
                [,COND=YES]
                [,NOSPACE=symbolic address]
                [,NORESP=symbolic address]
                [,IDERROR=symbolic address]
                [,IOERROR=symbolic address]
                [,INVREQ=symbolic address]
                [,ERROR=symbolic address]
```

```
        DFHTS   TYPE=PUTQ
                [,TYPOPER=REPLACE]
                [,DATAID=name]
                [,TSDADDR={symbolic address}]
                          {YES              }
                [,STORFAC={AUXILIARY}]
                          {MAIN      }
                [,ENTRY={n  }]
                        {YES}
                [,COND=YES]
                [,NOSPACE=symbolic address]
                [,NORESP=symbolic address]
                [,IOERROR=symbolic address]
                [,INVREQ=symbolic address]
                [,ENERROR=symbolic address]
                [,ERROR=symbolic address]
```

```
        DFHTS   TYPE=GET
                [,TYPOPER=EXCL]
                [,DATAID=name]
                [,TSDADDR={symbolic address}]
                          {YES              }
                [,RELEASE={YES}]
                          {NO }
                [,NORESP=symbolic address]
                [,IDERROR=symbolic address]
                [,IOERROR=symbolic address]
                [,INVREQ=symbolic address]
                [,ENERROR=symbolic address]
                [,ERROR=symbolic address]
```

```
        DFHTS   TYPE=GETQ
                [,TYPOPER=EXCL]
                [,DATAID=name]
                [,TSDADDR={symbolic address}]
                          {YES              }
                [,ENTRY={n  }]
                        {YES}
                [,NORESP=symbolic address]
                [,IDERROR=symbolic address]
```

```
|        |         | [,IOERROR=symbolic address]                                |
|        |         | [,INVREQ=symbolic address]                                 |
|        |         | [,ENERROR=symbolic address]                                |
|        |         | [,ERROR=symbolic address]                                  |
+--------+---------+------------------------------------------------------------+
|        | DFHTS   | TYPE=RELEASE                                               |
|        |         | [,DATAID=name]                                             |
|        |         | [,NORESP=symbolic address]                                 |
|        |         | [,IDERROR=symbolic address]                                |
|        |         | [,INVREQ=symbolic address]                                 |
|        |         | [,ERROR=symbolic address]                                  |
+--------+---------+------------------------------------------------------------+
|        | DFHTS   | TYPE=PURGE                                                 |
|        |         | [,DATAID=name]                                             |
|        |         | [,NORESP=symbolic address]                                 |
|        |         | [,IDERROR=symbolic address]                                |
|        |         | [,INVREQ=symbolic address]                                 |
|        |         | [,ERROR=symbolic address]                                  |
+--------+---------+------------------------------------------------------------+
|        | DFHTS   | TYPE=CHECK                                                 |
|        |         | [,NOSPACE=symbolic address]                                |
|        |         | [,NORESP=symbolic address]                                 |
|        |         | [,IDERROR=symbolic address]                                |
|        |         | [,IOERROR=symbolic address]                                |
|        |         | [,INVREQ=symbolic address]                                 |
|        |         | [,ENERROR=symbolic address]                                |
|        |         | [,ERROR=symbolic address]                                  |
+--------+---------+------------------------------------------------------------+
```

STORAGE SERVICES

```
+--------+---------+------------------------------------------------------------+
|        | DFHSC   | TYPE=GETMAIN                                               |
|        |         | [,INITIMG={number}]                                        |
|        |         |           {YES   }                                         |
|        |         | [,NUMBYTE=number]                                          |
|        |         | [,COND={YES                    }]                          |
|        |         |        {(YES,symbolic address)}                            |
|        |         |        {(NO,symbolic address) }                            |
|        |         | ,CLASS={TERMINAL or TERM}                                  |
|        |         |        {USER            }                                  |
|        |         |        {TRANSDATA or TD }                                  |
|        |         |        {TEMPSTRG or TS  }                                  |
+--------+---------+------------------------------------------------------------+
|        | DFHSC   | TYPE=FREEMAIN                                              |
|        |         | [,RELEASE=ALL]                                             |
+--------+---------+------------------------------------------------------------+
```

PROGRAM SERVICES

```
DFHPC   TYPE=LINK
        [,PROGRAM=name]
        [,COND=YES]
        [,NORESP=symbolic address]
        [,PGMIDER=symbolic address]

DFHPC   TYPE=XCTL
        [,PROGRAM=name]

DFHPC   TYPE=LOAD
        [,PROGRAM=name]
        [,LOADLST=NO]
        [,COND=YES]
        [,NORESP=symbolic address]
        [,PGMIDER=symbolic address]

DFHPC   TYPE=RETURN
        [,TRANSID=transaction code]

DFHPC   TYPE=DELETE
        [,PROGRAM=name]

DFHPC   TYPE=ABEND
        [,ABCODE={value}]
                 {YES  }
        [,CANCEL=YES]

DFHPC   TYPE=SETXIT
        [,PROGRAM={name}][,ROUTINE={symbolic address}]
                  {YES }           {YES              }
        [,NORESP=symbolic address]
        [,PGMIDER=symbolic address]

DFHPC   TYPE=RESETXIT

DFHPC   TYPE=COBADDR
        ,LABEL=symbolic label
```

```
DFHPC   TYPE=CHECK
       [,NORESP=symbolic address]
       [,PGMIDER=symbolic address]
```

TIME SERVICES

```
DFHIC   TYPE=GETIME
       [,FORM={BINARY|PACKED}]
       [,TIMADR={symbolic address|YES}]
       [,NORESP=symbolic address]
       [,INVREQ=symbolic address]
       [,ERROR=symbolic address]
```

```
DFHIC   TYPE=WAIT
       [,INTRVAL={numeric value|YES}]|[,TIME={numeric value|YES}]
       [,REQID={name|YES|'prefix'}]
       [,NORESP=symbolic address]
       [,INVREQ=symbolic address]
       [,EXPIRD=symbolic address]
       [,ERROR=symbolic address]
```

```
DFHIC   TYPE=POST
       [,INTRVAL={numeric value|YES}]|[,TIME={numeric value|YES}]
       [,REQID={name|YES|'prefix'}]
       [,NORESP=symbolic address]
       [,INVREQ=symbolic address]
       [,EXPIRD=symbolic address]
       [,ERROR=symbolic address]
```

```
DFHIC   TYPE=INITIATE
       [,INTRVAL={numeric value|YES}]|[,TIME={numeric value|YES}]
       [,REQID={name|YES|'prefix'}]
       [,TRANSID=name]
       [,TRMIDNT={name|YES}]
       [,NORESP=symbolic address]
       [,INVREQ=symbolic address]
       [,TRNIDER=symbolic address]
       [,TRMIDER=symbolic address]
       [,ERROR=symbolic address]
```

```
DFHIC   TYPE=PUT
        [,INTRVAL={numeric value}][,TIME= numeric value]
                  {YES          }        {      YES     }
        [,REQID={name    }]
                {YES     }
                {'prefix'}
        [,TRANSID=name]
        [,TRMIDNT={name}]
                  {YES }
        [,ICDADDR={symbolic address}]
                  {YES              }
        [,NORESP=symbolic address]
        [,INVREQ=symbolic address]
        [,TRNIDER=symbolic address]
        [,TRMIDER=symbolic address]
        [,IOERROR=symbolic address]
        [,ERROR=symbolic address]

DFHIC   TYPE=GET
        [,ICDADDR={symbolic address}]
                  {YES              }
        [,RELEASE=NO]
        [,NORESP=symbolic address]
        [,INVREQ=symbolic address]
        [,NOTFND=symbolic address]
        [,ENDDATA=symbolic address]
        [,IOERROR=symbolic address]
        [,TSINVLD=symbolic address]
        [,ERROR=symbolic address]

DFHIC   TYPE=CANCEL
        [,REQID={name}]
                {YES }
        [,NORESP=symbolic address]
        [,INVREQ=symbolic address]
        [,NOTFND=symbolic address]
        [,ERROR=symbolic address]

DFHIC   TYPE=RETRY
        [,RELEASE=NO]
        [,NORESP=symbolic address]
        [,INVREQ=symbolic address]
        [,NOTFND=symbolic address]
        [,IOERROR=symbolic address]
        [,ERROR=symbolic address]
```

```
        DFHIC   TYPE=CHECK
                [,NORESP=symbolic address]
                [,INVREQ=symbolic address]
                [,EXPIRD=symbolic address]
                [,TRNIDER=symbolic address]
                [,TRMIDER=symbolic address]
                [,NOTFND=symbolic address]
                [,ENDDATA=symbolic address]
                [,IOERROR=symbolic address]
                [,ERROR=symbolic address]
                [,TSINVLD=symbolic address]
```

TASK SERVICES

```
        DFHKC   TYPE=ATTACH
                [,FCADDR=symbolic address]
                [,TRANSID=name]

        DFHKC   TYPE=CHAP
                [,PRTY=priority value]

        DFHKC   TYPE=WAIT
                ,DCI={SINGLE}
                     {LIST  }
                     {DISP  }
                [,ECADDR=symbolic address]

        DFHKC   TYPE=ENQ
                [,QARGADR=symbolic address]
                [,QARGLNG=number]

        DFHKC   TYPE=DEQ
                [,QARGADR=symbolic address]
                [,QARGLNG=number]

        DFHKC   TYPE=PURGE

        DFHKC   TYPE=NOPURGE
```

```
DFHJC   TYPE=GETJCA
```

```
DFHJC   TYPE={PUT          }
              {(WRITE,WAIT)}
        ,JFILEID={nn    }
                 {SYSTEM}
                 {YES   }
        [,JTYPEID={nnnn}]
                  {YES }
        [,JCDADDR={symbolic address}]
                  {YES              }
        [,JCDLGTH={decimal value}]
                  {YES           }
        [,PFXADDR={symbolic address}]
                  {YES              }
        [,PFXLGTH={decimal value}]
                  {YES           }
        [,NORESP=symbolic address]
        [,IDERROR=symbolic address]
        [,LERROR=symbolic address]
        [,IOERROR=symbolic address]
        [,NOTOPEN=symbolic address]
        [,INVREQ=symbolic address]
```

```
DFHJC   TYPE=WRITE
        ,JFILEID={nn    }
                 {SYSTEM}
                 {YES   }
        [,JTYPEID={nnnn}]
                  {YES }
        [,JCDADDR={symbolic address}]
                  {YES              }
        [,JCDLGTH={decimal value}]
                  {YES           }
        [,PFXADDR={symbolic address}]
                  {YES              }
        [,PFXLGTH={decimal value}]
                  {YES           }
        [,STARTIO={YES}]
                  {NO }
        [,COND={(YES,symbolic address)}]
               {NO                    }
        [,NORESP=symbolic address]
        [,IDERROR=symbolic address]
        [,LERROR=symbolic address]
        [,NOTOPEN=symbolic address]
        [,INVREQ=symbolic address]
```

```
        DFHJC   TYPE=WAIT
                ,JFILEID={nn    }
                         {SYSTEM}
                         {YES   }
                [,NORESP=symbolic address]
                [,IDERROR=symbolic address]
                [,IOERROR=symbolic address]
                [,NOTOPEN=symbolic address]
                [,INVREQ=symbolic address]

        DFHJC   TYPE=CHECK
                [,NORESP=symbolic address]
                [,IDERROR=symbolic address]
                [,LERROR=symbolic address]
                [,IOERROR=symbolic address]
                [,NOTOPEN=symbolic address]
                [,INVREQ=symbolic address]
```

RESTART/RECOVERY SERVICES

```
        DFHSP   TYPE=USER
```

```
DFHTR  TYPE=ON
      [,STYPE={SINGLE                          }]
              {ALL                             }
              {(system symbol1[,sys... ])      }
              {SYSTEM                          }
              {USER                            }
              {FE                              }

DFHTR  TYPE=OFF
      [,STYPE={SINGLE                          }]
              {ALL                             }
              {(system symbol1[,sys... ])      }
              {SYSTEM                          }
              {USER                            }
              {FE                              }

DFHTR  TYPE=ENTRY
      [,STYPE={SYSTEM}]
              {USER  }
              {FE    }
       ,ID=number
      [,DATA1={symbol  }]   [,RDATA1={register  }]
              {(symbol)}            {(register)}
      [,DATA2={symbol  }]   [,RDATA2={register  }]
              {(symbol)}            {(register)}
      [,DATA1TP={HBIN   }]
                {FBIN   }
                {CHAR   }
                {PACK   }
                {POINTER}
      [,DATA2TP={HBIN   }]
                {FBIN   }
                {CHAR   }
                {PACK   }
                {POINTER}
```

```
        DFHDC   TYPE=TRANSACTION
                ,DMPCODE={value}
                         {YES  }

        DFHDC   TYPE=CICS
                ,DMPCODE={value}
                         {YES  }

        DFHDC   TYPE=COMPLETE
                ,DMPCODE={value}
                         {YES  }

        DFHDC   TYPE=PARTIAL
                LIST=([TERMINAL][,PROGRAM][,TRANSACTION][,SEGMENT])
                ,DMPCODE={value}
                         {YES  }
```

BUILT-IN FUNCTIONS

```
DFHBFTCA [OPTION={BASIC}]
         [       {WTRET}]
```

TABLE SEARCH

```
DFHBIF   TYPE=TSEARCH
        [,ARG=symbolic address]
        [,TARGET=symbolic address]
        [,ATABLE=([symbolic address1][,{symbolic address2}]
                                       {YES             }
                  [,numeric value1][,{numeric value2}]
                                     {YES           }
                  [,numeric value3])
        [,FTABLE=([{symbolic address1}] [,{symbolic address2}]
                  [{YES               }]  [{YES               }]
                  [,{numeric value1}] [,{numeric value2}])
                  [{YES           }]  [{YES           }]
        [,ORDER={ASCENDING }]
                {DESCENDING}
        [,SUBST={symbolic address}][,NOMATCH=symbolic address]
                {'literal value' }
        [,INDEX=symbolic address]
        [,RANGE=YES]
        [,ERROR=symbolic address]
```

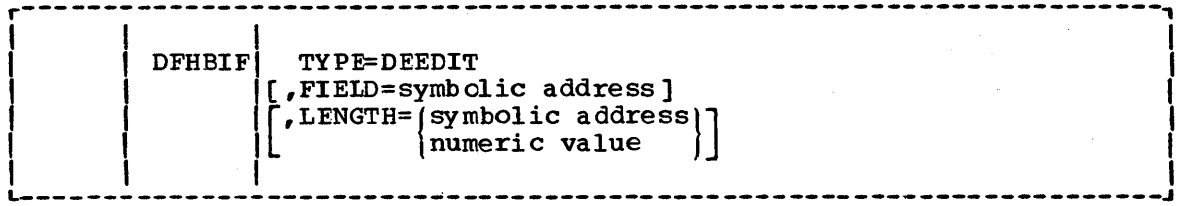
PHONETIC CONVERSION

```
DFHBIF   TYPE=PHONETIC
        [,FIELD=symbolic address]
        [,ERROR=symbolic address]
```

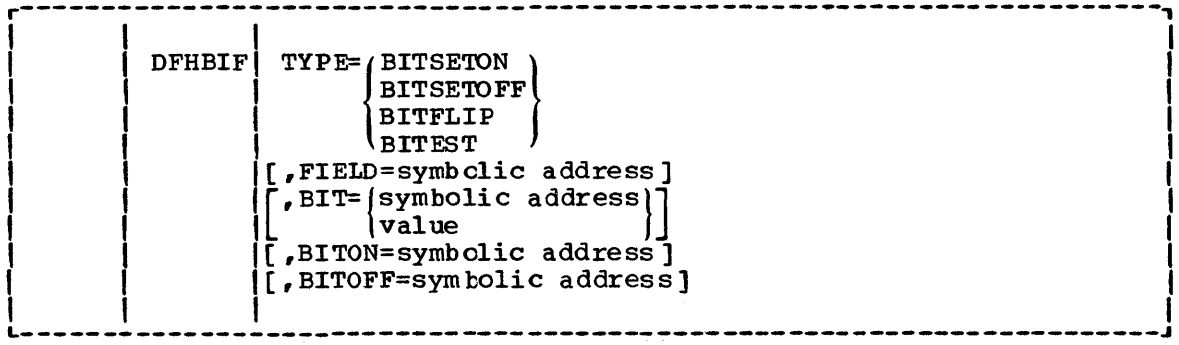
FIELD VERIFY

```
DFHBIF   TYPE=FVERIFY
        [,FIELD=symbolic address]
        [,LENGTH={symbolic address}]
                 {numeric value   }
        [,ALPHA=symbolic address]
        [,NUMERIC=symbolic address]
        [,PACKED=symbolic address]
```

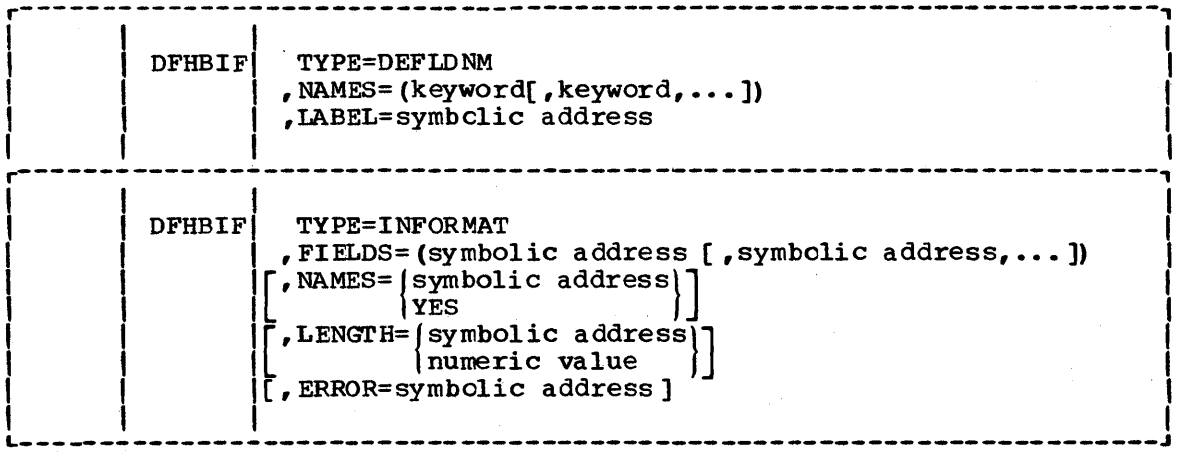
FIELD EDIT



BIT MANIPULATION



INPUT FORMATTING



WEIGHTED RETRIEVAL

```
DFHBIF   TYPE=WTRETST
        [,DATASET=symbolic name]
        [,RDIDADR=symbolic address]
        [,INPUTNO={symbolic address}]
                  {numeric value   }
                  {YES             }
        [,INPUTST={symbolic address}]
                  {numeric value   }
                  {YES             }
        [,INPUTPC=([suboperand1][,suboperand2])]
        [,NRECDS={symbolic address}]
                 {numeric value   }
                 {YES             }
        [,NORESP=symbolic address]
        [,DSIDER=symbolic address]
        [,NOTOPEN=symbolic address]
        [,NOTFND=symbolic address]
        [,INVREQ=symbolic address]
        [,IOERROR=symbolic address]
        [,OFLOW=symbolic address]
        [,ILLOGIC=symbolic address]

DFHBIF   TYPE=WTRTPARM
        [,FIELD1=([symbolic address][,numeric value][,char])]
        [,FIELD2=([symbolic address1][,symbolic address2])]
        [,NULL={symbolic address}]
               {character value }
               {YES             }
        [,MATCH={symbolic address}]
                {numeric value   }
        [,NOMATCH={symbolic address}]
                  {numeric value   }
        [,RANGE=(suboperand1,suboperand2[,suboperand3])]

DFHBIF   TYPE=WTRETGET
        [,NORESP=symbolic address]
        [,ENDFILE=symbolic address]
        [,NOTOPEN=symbolic address]
        [,NOTFND=symbolic address]
        [,INVREQ=symbolic address]
        [,IOERROR=symbolic address]
        [,OFLOW=symbclic address]
        [,ILLOGIC=symbolic address]

DFHBIF   TYPE=WTRETREL
        [,NORESP=symbolic address]
        [,INVREQ=symbolic address]
        [,ILLOGIC=symbolic address]
```

```
        |          |
        | DFHBIF   |  TYPE=WTRETCHK
        |          |[,NORESP=symbolic address]
        |          |[,DSIDER=symbolic address]
        |          |[,NOTOPEN=symbolic address]
        |          |[,NOTFND=symbolic address]
        |          |[,INVREQ=symbolic address]
        |          |[,ENDFILE=symbolic address]
        |          |[,IOERROR=symbolic address]
        |          |[,OFLOW=symbolic address]
        |          |[,ILLOGIC=symbolic address]
        |          |
```

BASIC MAPPING SUPPORT SERVICES

```
        |          |
 mapset | DFHMSD   |[TYPE={DSECT}]
        |          |     {MAP  }
        |          |     {FINAL}
        |          |[,MODE={IN   }]
        |          |      {OUT  }
        |          |      {INOUT}
        |          |[,LANG={ASM  }]
        |          |       {COBOL}
        |          |       {PLI  }
        |          |[,BASE=name]
        |          |[,TERM=terminal type]
        |          |[,CTRL=([PRINT],[{L40    }][,FREEKB][,ALARM][,FRSET])
        |          |               {L64    }
        |          |               {L80    }
        |          |               {HONEOM }
        |          |[,DATA={FIELD}]
        |          |       {BLOCK}
        |          |[,OBFMT={YES}]
        |          |        {NO }
        |          |[TIOAPFX={YES}]
        |          |         {NO }
        |          |
        |          |
 map    | DFHMDI   |[SIZE=(line,column)]
        |          |[,LINE={number}]
        |          |       {NEXT  }
        |          |       {SAME  }
        |          |[,COLUMN={number}]
        |          |         {NEXT  }
        |          |         {SAME  }
        |          |[,JUSTIFY=([{LEFT }][,{FIRST}])
        |          |           {RIGHT}   {LAST }
        |          |[,HEADER=YES]
        |          |[,TRAILER=YES]
        |          |[,DATA={FIELD}]
        |          |       {BLOCK}
        |          |[,OBFMT={YES}]
        |          |        {NO }
        |          |[TIOAPFX={YES}]
        |          |         {NO }
        |          |
```

```
[fld]  DFHMDF  [,POS={number|(line,column)}]
               [,ATTRB=([{ASKIP|PROT|UNPROT}] [,NUM] [,{BRT|NORM|DRK}] [,DET][,IC][,FSET])]
               [,LENGTH=number]
               [,JUSTIFY=([{LEFT|RIGHT}][,{BLANK|ZERO}])]
               [,INITIAL='any user information']
               [,GRPNAME=group name]
               [,OCCURS=number]
               [,PICIN='value']
               [,PICOUT='value']

       DFHBMS  TYPE={IN|MAP} [,SAVE][,TEXT])
               [,MAP={map name|YES}] | [,MAPADR={symbolic address|YES}]
               [,MAPSET={map set name|YES}] [,MSETADR={symbolic address|YES}]
               [,RDATT=symbolic address]
               [,NORESP=symbolic address]
               [,MAPFAIL=symbolic address]
               [,INVMPSZ=symbolic address]
               [,ERROR=symbolic address]

       DFHBMS  TYPE=(PAGEBLD [,{OUT[,WAIT]|STORE|RETURN}] [,SAVE][,ERASE]
                     [,LAST])
               [,IOTYPE={IMMED|DELAY}]
               [,LDC={mnemonic|YES}]
               [,DATA={NO|YES|ONLY}]
               [,MAP={map name|YES}]
               [,MAPSET={mapset name|YES}] | [,MSETADR={symbolic address|YES}]
               [,CTRL=([PRINT] [,{L40|L64|L80|HONEOM}] [,FREEKB][,ALARM][,FRSET])]
               [,OFLOW=symbolic address]
               [PROPT=NLEOM]
               [,CURSOR={number|YES}]
               [,REQID={'prefix'|YES}]
               [,FMHPARM={parameter|YES}]
               [,WRBRK=symbolic address]          CICS/OS/VS only
               [,NORESP=symbolic address]
               [,TSIOERR=symbolic address]
               [,INVREQ=symbolic address]
               [,RETPAGE=symbolic address]
               [,INVLDC=symbolic address]
               [,INVMPSZ=symbolic address]
               [,IGREQID=symbolic address]
               [,ERROR=symbolic address]
```

```
DFHBMS  TYPE=(TEXTBLD [,{OUT[,WAIT]}]    [,SAVE][,ERASE]
                        {STORE      }
                        {RETURN     }
              [,LAST])
        [,IOTYPE={IMMED}]
                 {DELAY}
        [,LDC={mnemonic}]
              {YES     }
        [,HEADER={symbolic address}]
                 {YES              }
        [,TRAILER={symbolic address}]
                  {YES              }
        [,JUSTIFY={FIRST}]
                  {LAST }
                  {nnn  }
                  {YES  }
        [,CTRL=([PRINT][,{L40   }][,FREEKB][,ALARM])
                         {L64   }
                         {L80   }
                         {HONEOM}
        [PROPT=NLEOM]
        [,CURSOR={number}]
                 {YES   }
        [,REQID={'prefix'}]
                {YES      }
        [,FMHPARM={parameter}]
                  {YES      }
        [,WRBRK=symbolic address]          CICS/OS/VS only
        [,NORESP=symbolic address]
        [,TSIOERR=symbolic address]
        [,INVREQ=symbolic address]
        [,INVLDC=symbolic address]
        [,RETPAGE=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbolic address]
```

```
DFHBMS  TYPE=([{OUT[,WAIT]}]   [,NOEDIT][,SAVE][ERASE]
               [{STORE    }]
               [{RETURN   }]
               [,LAST])
        [,IOTYPE={IMMED}]
                 {DELAY}
        [,LDC={mnemonic}]
              {YES     }
        [,DATA={NO  }]
               {YES }
               {ONLY}
        [,MAP={map name}] | [,MAPADR={symbolic address}]
              {YES     }            {YES              }
        [,MAPSET={mapset name}] | [,MSETADR={symbolic address}]
                 {YES        }              {YES              }
        [,CTRL=([PRINT][,{L40   }][,FREEKB][,ALARM][,FRSET])]
                         {L64   }
                         {L80   }
                         {HONEOM}
        [PROPT=NLEOM]
        [,CURSOR={number}]
                 {YES   }
        [,REQID={'prefix'}]
                {YES     }
        [,FMHPARM={parameter}]
                  {YES      }
        [,WRBRK=symbolic address]  CICS/OS/VS only
        [,NORESP=symbolic address]
        [,TSIOERR=symbolic address]
        [,INVREQ=symbolic address]
        [,INVLDC=symbolic address]
        [,RETPAGE=symbolic address]
        [,INVMPSZ=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbolic address]

DFHBMS  TYPE=(PAGEOUT[,LAST])
        [,CTRL=([{PAGE    }]   [,{RETAIN }])]
                 {AUTOPAGE}     {RELEASE}
        [,TRAILER={symbolic address}]
                  {YES             }
        [,TRANSID=transaction code]
        [,WRBRK={symbolic address}]CICS/OS/VS only
                {CURRENT        }
                {ALL            }
        [,EODPURG={AUTO}]
                  {OPER}
        [,REQID={'prefix'}]
                {YES     }
        [,FMHPARM={parameter}]
                  {YES      }
        [,NORESP=symbolic address]
        [,TSIOERR=symbolic address]
        [,RETPAGE=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbclic address]

DFHBMS  TYPE=PURGE
```

```
DFHBMS  TYPE=ROUTE
        [,LDC={mnemonic}]
              {YES     }
        [,LIST={symbolic address}]
               {YES             }
               {ALL             }
        [,INTRVAL={numeric value}] | [,TIME={numeric value}]
                  {YES           }         {YES           }
        [,OPCLASS={decimal value,...}]
                  {YES               }
        [,TITLE={symbolic address}]
                {YES             }
        [,ERRTERM={termid}]
                  {ORIG  }
                  {YES   }
        [PROPT=NLEOM]
        [,REQID={'prefix'}]
                {YES     }
        [,NORESP=symbolic address]
        [,INVET=symbolic address]
        [,RTEFAIL=symbolic address]
        [,RTESOME=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbolic address]

DFHBMS  TYPE=CHECK
        [,NORESP=symbolic address]
        [,TSIOERR=symbolic address]
        [,INVREQ=symbolic address]
        [,RETPAGE=symbolic address]
        [,MAPFAIL=symbolic address]
        [,INVET=symbolic address]
        [,INVLDC=symbolic address]
        [,RTEFAIL=symbolic address]
        [,RTESOME=symbolic address]
        [,INVMPSZ=symbolic address]
        [,IGREQID=symbolic address]
        [,ERROR=symbolic address]
```

CICS/VS-DL/I INTERFACE SERVICES

```
DFHFC   TYPE=(DL/I,PCB)
        [PSB={'psbname'}]
             {symbolic address}
             {YES        }
        [,NORESP=symbolic address]
        [,DLINA=symbolic address]
        [,PSBSCH=symbolic address]
        [,PSBNF=symbolic address]
        [,PSBFAIL=symbolic address]
        [,INVREQ=symbolic address]
```

```
DFHFC   TYPE=(DL/I [,function])
        [,PCB={symbolic address}]
              {(register)      }
        [,WRKAREA={symbolic address}]
                  {YES             }
                  {(register)      }
        [,SSAS={NO                                    }]
               {([ssacount][,ssa1][,ssa2,...])        }
               {([(register1)][,(register2),...])     }
        [,SSALIST={YES                }]
                  {NO                 }
                  {symbolic address   }
                  {(register)         }
        [,NORESP=symbolic address]
        [,NOTOPEN=symbolic address]
        [,DLINA=symbolic address]
        [,FUNCNS=symbolic address]
        [,INVREQ=symbolic address]
```

```
DFHFC   TYPE=(DL/I,{TERM})
                   {T   }
        [,DLINA=symbolic address]
        [,TERMNS=symbolic address]
        [,INVREQ=symbolic address]
```

```
DFHFC   TYPE=CHECK
        [,NORESP=symbolic address]
        [,DLINA=symbolic address]
        [,PSBSCH=symbolic address]
        [,PSBNF=symbolic address]
        [,PSBFAIL=symbolic address]
        [,FUNCNS=symbolic address]
        [,TERMNS=symbolic address]
        [,LANGCON=symbolic address]      CICS/DOS/VS only
        [,TASKNA=symbolic address]       CICS/DOS/VS only
        [,PSBNA=symbolic address]        CICS/DOS/VS only
        [,INVREQ=symbolic address]
        [,NOTOPEN=symbolic address]
```

APPENDIX E. TRANSLATE TABLES FOR THE 2980

This section contains translate tables for the following components of the 2980 General Banking Terminal System:

1. 2980 Teller Station Model 1
2. 2980 Administrative Station Model 2
3. 2980 Teller Station Model 4

The line codes and CPU codes listed in these tables are unique to CICS/VS and are represented as standard EBCDIC characters.

2980-1 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
0	MSG ACK	1	↕	1	F1	AA	F1	1
1	SEND AGAIN	Q	R	Q	D8	D9	D8	
2	CORR	A	C	A	C1	C3	C1	
3	HOLD OVERRIDE	2	H	2	F2	C8	F2	
4	VOID	Z	V	Z	E9	E5	E9	
5	ACCT INQ	W	Q	W	E6	D8	E6	
6	ACCT TFR	S	T	S	E2	AB	E2	2
7	CIF	3	¢	3	F3	AC	F3	3
8	MISC	X	¬	X	E7	AD	E7	4
9	CLSD ACCT	E	X	E	C5	E7	C5	
10	NO BOOK	D	¦	D	C4	AE	C4	5
11	MORT LOAN	4	M	4	F4	AF	F4	6
12		C	+	C	C3	B0	C3	7
13	NEW ACCT	R	A	R	D9	B1	D9	8
14	BOOK BAL	F	§	F	C6	B2	C6	9
15	INST LOAN	5	L	5	F5	B3	F5	10
16	SPEC TRAN	V	$	V	E5	B4	E5	11
17	SAV BOND	T	B	T	E3	B5	E3	12

2980-1 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
18	SAV	G	S	G	C7	B6	C7	13
19	XMAS CLUB	6	C	6	F6	B7	F6	14
20	.	B	.	B	C2	4B	C2	
21	DDA	Y	D	Y	E8	B8	E8	15
22	@	H	@	H	C8	B9	C8	16
23	MON ORD	7	M	7	F7	BA	F7	17
24	0	N	0	N	D5	F0	D5	
25	7	U	7	U	E4	F7	E4	
26	4	J	4	J	D1	F4	D1	
27	CSHR CHK	8	I	8	F8	BB	F8	18
28	1	M	1	M	D4	F1	D4	
29	8	I	8	I	C9	F8	C9	
30	5	K	5	K	D2	F5	D2	
31	CASH RECD	9	C	9	F9	BC	F9	19
32	2	,	2	,	6B	F2	6B	
33	9	0	9	0	D6	F9	D6	
34	6	L	6	L	D3	F6	D3	

2980-1 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
35	UTIL BILL	0	U	0	F0	E4	F0	
36	3	.	3	.	4B	F3	4B	
37	DEP +	P	+	P	D7	4E	D7	
38	WITH −	$	−	$	5B	60	5B	
39	FEES	−	F	−	60	C6	60	
40	TOTL	/	T	/	61	E3	61	
41	CASH IN	*	¢	*	5C	BD	5C	20
42	CASH CHK	#	¢	#	7B	BE	7B	21
43	VAL	&	A-K	&	50	STATION ID	50	
44	TAB				05	05	05	TABCHAR
45	ALPHA ENTRY				36			
46	NUMERIC ENTRY				06			
47	SEND				26-ETB 03-ETX			
48	RETURN				15	15	15	JRNLCR
49	NUMERIC ENTRY				06			
50	SPACE				40	40	40	
58	MSGLIGHT				17	17	17	MSGLITE

2980-2 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
0	= 1		1	=	F1	F1 (1)	7E (=)	
1	Q		q	Q	D8	98 (q)	D8 (Q)	
2	A		a	A	C1	81 (a)	C1 (A)	
3	2		2	<	F2	F2 (2)	4C (<)	
4	Z		z	Z	E9	A9 (z)	E9 (Z)	
5	W	NO KEYFRONT ENGRAVING	w	W	E6	A6 (w)	E6 (W)	
6	S		s	S	E2	A2 (s)	E2 (S)	
7	; 3		3	;	F3	F3 (3)	5E (;)	
8	X		x	X	E7	A7 (x)	E7 (X)	
9	E		e	E	C5	85 (e)	C5 (E)	
10	D		d	D	C4	84 (d)	C4 (D)	
11	: 4		4	:	F4	F4 (4)	7A (:)	
12	C		c	C	C3	83 (c)	C3 (C)	
13	R		r	R	D9	99 (r)	D9 (R)	
14	F		f	F	C6	86 (f)	C6 (F)	
15	% 5	NO KEYFRONT ENGRAVING	5	%	F5	F5 (5)	6C (%)	
16	V		v	V	E5	A5 (v)	E5 (V)	
17	T		t	T	E3	A3 (t)	E3 (T)	
18	G		g	G	C7	87 (g)	C7 (G)	
19	' 6		6	'	F6	F6 (6)	7D (')	
20	B		b	B	C2	82 (b)	C2 (B)	
21	Y		y	Y	E8	A8 (y)	E8 (Y)	
22	H		h	H	C8	88 (h)	C8 (H)	
23	> 7		7	>	F7	F7 (7)	6E (>)	
24	N		n	N	D5	95 (n)	D5 (N)	
25	U		u	U	E4	A4 (u)	E4 (U)	

2980-2 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
26	J		j	J	D1	91 (j)	D1 (J)	
27	* 8		8	8	F8	F8 (8)	5C (*)	
28	M		m	M	D4	94 (m)	D4 (M)	
29	I		i	I	C9	89 (i)	C9 (I)	
30	K		k	K	D2	92 (k)	D2 (K)	
31	(9		9	(	F9	F9 (9)	4D (()	
32	| ,		,	|	6B	6B (,)	4F (|)	
33	O		o	O	D6	96 (o)	D6 (O)	
34	L		l	L	D3	93 (l)	D3 (L)	
35	) 0		0	)	F0	F0 (0)	5D ())	
36	¬ .	NO KEYFRONT ENGRAVING	.	¬	4B	4B (.)	5F (¬)	
37	P		p	P	D7	97 (p)	D7 (P)	
38	! $		$	!	5B	5B ($)	5A (!)	
39	_ -		-	_	60	60 (-)	6D (_)	
40	? /		/	?	61	61 (/)	6F (?)	
41	¢ @		@	¢	5C	7C (@)	4A (¢)	
42	= #		#	"	7B	7B (#)	7F (")	
43	+ &		&	+	50	50 (&)	4E (+)	
44	TAB				05	05	05	
45	LOCK	NO KEYFRONT ENGRAVING			36	36	36	
46	SHIFT				06	06	06	
47	BACKSPACE				16	16	16	BCKSPACE
48	RETURN				15	15	15	
49	SHIFT				06	06	06	
50	(SPACE)				40	40	40	
53	SEND				26-ETB 03-ETX			

2980-4 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
0	CK $	-	¢	-	D9	BC	60	19
1		Q	L	Q	D3	D3	D8	
2		A	A	A	C1	C1	C1	
3	CK #	0	$\bar{C}$	0	C9	B7	C9	14
4		Z	.	Z	E9	4B	E9	
5		W	*	W	E6	5C	E6	
6		S	$	S	E2	5B	E2	
7	IMD 2	1	!	1	5B	4F	F1	
8		X	¤	X	E7	AE	E7	5
9		E	E	E	C5	C5	C5	
10		D	?	D	C4	6F	C4	
11	IMD 1	2	M	2	4B	D4	F2	
12		C	C	C	C3	C3	C3	
13		R	-	R	60	60	D9	
14		F	F	F	C6	C6	C6	
15	CODE	3	I	3	E8	BB	F3	18
16		V	▽	V	E5	A0	E5	22
17		T	△	T	E3	A1	E3	23

2980-4 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
18		G	G	G	C7	C7	C7	
19	AMT	4	$	4	5C	BE	F4	21
20		B	B	B	C2	C2	C2	
21		Y	/	Y	61	61	E8	
22		H	P	H	D7	D7	C8	
23	OB	5	%	5	D8	B2	F5	9
24		N	N	N	D5	D5	D5	
25		U	M	U	E4	AF	E4	6
26		J	J	J	D1	D1	D1	
27	ACCT #	6	#	6	C8	7B	F6	
28		N	X	M	D4	E7	D4	
29		I	0	I	D6	D6	C9	
30		K	K	K	D2	D2	D2	
31	7	7	7	7	F7	F7	F7	
32	---	---	,	,	6B	BLANK	6B	
33	4	0	4	0	F4	F4	D6	
34	1	L	1	L	F1	F1	D3	

2980-4 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
35	8	8	8	8	F8	F8	F8	
36	0	.	0	.	F0	F0	4B	
37	5	P	5	P	F5	F5	D7	
38	2	$	2	$	F2	F2	5B	
39	9	9	9	9	F9	F9	F9	
40	---	---	+	*	7B	BO	7B	7
41	6	*	6	*	F6	F6	5C	
42	3	#	3	#	F3	F3	7B	
43	VAL	&	A-K	&	50	50	50	
44	TAB				05	05	05	
45	ALPHA				36			
46	NUMERIC				06			
47	SEND				26-ETB 03-ETX			
48	RETURN				15	15	15	
49	NUMERIC				06			
50	SPACE				40	40	40	
51	FEED OPEN				04			OPENCH

BIBLIOGRAPHY

For further information concerning the Customer Information Control System/Virtual Storage (CICS/VS), the reader of this manual is referred to the following IBM CICS/VS publications:

Customer Information Control System/Virtual Storage (CICS/VS):

- General Information Manual (GH20-1280)
- Installation Guide - CICS/DOS/VS (SH20-9051)
- System/Application Design Guide (SH20-9002)
- System Programmer's Reference Manual (SH20-9004)
- Advanced Communication Guide (SH20-9049)
- Terminal Operator's Guide (SH20-9005)
- System Administrator's Guide (SH20-9006)
- Messages and Codes Manual (SH20-9008)
- Operations Guide (OS/VS) (SH20-9011)
- Operations Guide (DOS/VS) (SH20-9012)
- Program Logic Manual (OS/VS) (LY20-8006)
- Program Logic Manual (DOS/VS) (LY20-8007)

A summary of CICS/VS debugging information is provided in CICS/VS Reference Summary: Program Debugging (SX26-3701).

A summary of CICS/VS master-terminal commands is provided in the CICS/VS Reference Summary: Master Terminal Operator (SX26-3700).

The reader of this manual may also want to refer to the following IBM publications:

IBM System/360 Disk Operating System:

- Subset American National Standard COBOL Compiler and Library Programmer's Guide (SC28-6439)
- Full American National Standard COBOL Compiler and Library, Version 3, Programmer's Guide (SC28-6441)
- Full American National Standard COBOL Programmer's Guide (GC28-6398)

IBM System/360 Operating System:

- Full American National Standard COBOL Compiler and Library, Version 4, Programmer's Guide (SC28-6456)
- Full American National Standard COBOL Compiler and Library, Version 3, Programmer's Guide (SC28-6437)
- Full American National Standard COBOL Compiler and Library, Version 2, Programmer's Guide (GC28-6399)

DOS/VS COBOL Compiler and Library Programmer's Guide (SC28-6478)
OS/VS COBOL Compiler and Library Programmer's Guide (SC28-6483)
OS PL/I Optimizing Compiler Programmer's Guide (SC33-0006)
OS DOS PL/I Optimizing Compiler Programmer's Guide (SC33-0008)
IBM System/360 Operating System PL/I (F) Programmer's Guide (GC28-6594)

IMS/VS Application Programming Reference Manual (SH20-9026)

DL/I DOS/VS Application Programming Reference Manual (SH12-5411)
DL/I DOS/VS Utilities and Guide for the System Programmer (SH12-5412)
IBM 3270 Information Display System Component Description (GA27-2749)
Component Description: IBM 2721 Portable Audio Terminal (GA27-3029)
IBM 2780 Data Transmission Terminal Component Description (GA27-3035)

AVAILABILITY OF PUBLICATIONS

The availability of a publication is indicated by its use key, which is the first letter in the order number. The use keys and their meanings are:

G -- Generally available: Provided to users of IBM systems, products, and services without charge, in quantities to meet their normal requirements. Can also be purchased by anyone through IBM branch offices.

S -- Sold: Can be purchased by anyone through IBM offices.

L -- Licensed material, property of IBM: Available only to licensees of the related program products under the terms of the license agreements.

ABCODE operand 223
ABEND
  code 223
  operand 223
  TCA field 26
  termination code 26
Abnormal termination
  activate ABEND exit 126
  cancel ABEND exit 126
  code 223
  reactivate ABEND exit 126
  transaction 125
Addressability
  application program 12
  BMS operation 360
  common system area (CSA)
    ANS COBOL 40
    assembler language 31
    PL/I 51
  common work area (CWA)
    ANS COBOL 40
    assembler language 31
    PL/I 51
  file input/output area (FIOA)
    ANS COBOL 42
    assembler language 33
    PL/I 53
  file work area (FWA)
    ANS COBOL 42
    assembler language 34
    PL/I 54
  journal control area (JCA)
    ANS COBOL 45
    assembler language 37
    PL/I 56
  storage accounting area (SAA)
    ANS COBOL 44
    assembler language 36
    PL/I 56
  storage area 21
  task control area (TCA)
    ANS COBOL 41
    assembler language 32
    PL/I 52
  temporary storage input/output area (TSIOA)
    ANS COBOL 44
    assembler language 36
    PL/I 55
  terminal control table terminal entry (TCTTE)
    ANS COBOL 40
    assembler language 31
    PL/I 52
  terminal input/output area (TIOA)
    ANS COBOL 41
    assembler language 33
    PL/I 52
  transaction work area (TWA)
    ANS COBOL 41
    assembler language 32
Addressability (Continued)
  transaction work area (TWA) (Continued)
    PL/I 52
  transient data input area (TDIA)
    ANS COBOL 43
    assembler language 35
    PL/I 54,54
  transient data output area (TDOA)
    ANS COBOL 43
    assembler language 35
    PL/I 55
  VSAM work area (VSWA)
    ANS COBOL 43
    assembler language 34
    PL/I 54
ALPHA operand 311
ANS COBOL
  addressability of storage areas 21
  COPY statement 20
  data constant location 39
  DEPENDING ON clause usage 46
  guidelines 45
  link-editing 15
  multipunch codes 94
  program example
    abnormally terminate a transaction 125
    acquire the journal control area 157
    asynchronous journal output 162
    automatic task initiation 138
    BMS examples 405
    BMS map definition 355
    BMS response codes 398
    BMS TIOA specification 403
    browse operation 85
    building a new record 79
    change priority of a task 148
    copying storage definitions 47
    copying storage referred by BIF 302
    delete a program 124
    DL/I requests 464
    establish program exit 126
    examine TCTTETAB 426
    executable CICS/VS sample program 475
    file control response codes 94
    forced end of volume 104
    freeing temporary storage 113
    journal record synchronization 165
    linking programs 121
    loading a program 123
    multiple events task synchronization 151
    obtain and initialize main storage 117
    obtaining a FWA 79
    program control response codes 128
    random read-only operation 71
    random retrieval for update 73
    random retrieval via indirect access 74
    random update or add data 77

ANS COBOL (Continued)
  program example (Continued)
    read variable-length record 102
    release main storage 118
    releasing a FWA 81
    relinquish control to higher priority task 152
    reset sequential retrieval 91
    retrieval of time-ordered data 140
    retrieve temporary data 111
    signal for time expired 135
    single event task synchronization 150
    single-server resource synchronization 153,154
    store temporary data 110
    suspend task processing 133
    synchronous journal output 159
    temporary storage response codes 114
    terminal services 66
    terminate sequential retrieval 90
    time of day services 131
    time-ordered task initiation 137,138
    time service response codes 144
    transfer of program control 122
    transient data response codes 107
    VSAM locate mode I/O 72
    weighted retrieval 330
    write data - predefined symbolic destination 100
  register usage 13
  restrictions 14
  special compiler control statements 47
  storage definitions 39
  transfer of control 13
  variable data location 39
  WAIT requirement 172
  work area size 46
Appendix A. 469
Appendix B. 481
Appendix C. 485
Application programs
  addressability 13
  basic characteristics 7
  CICS/VS macro instruction 10
  coding aids 12
  communication and logical relationships 120
  considerations of virtual storage 15
  deleting a 124
  general structure 7
  initialization 13
  languages 6
  link-editing 15
  loading an 122
  packaging 9
  programming considerations 415
  programming techniques 7
  quasi-reentrance 9
  reference pattern 9
  register usage 13
  restrictions 14,15
  system environment 6
  techniques 15
  testing and debugging 251
  transfer of control 13
ARG operand 303
ARGTYP operand
  DFHFC TYPE=DELETE 193
  DFHFC TYPE=GET 189
  DFHFC TYPE=GETAREA 195
  DFHFC TYPE=PUT 191
  DFHFC TYPE=RESETL 201
  DFHFC TYPE=SETL 197
Assembler language
  addressability of storage areas 21
  addressability requirement 13
  COPY statement 20
  link-editing 15
  program example
    abnormally terminate a transaction 125
    acquire the journal control area 157
    asynchronous journal output 162
    automatic task initiation 138
    BMS examples 405
    BMS map definition 355
    BMS response codes 398
    BMS TIOA specification 403
    browse operation 85
    building a new record 79
    change priority of a task 148
    copying storage definitions 37
    copying storage referred by BIF 302
    delete a program 124
    DL/I requests 462
    establish program exit 126
    examine TCTTETAB 426
    executable CICS/VS sample programs 469
    file control response codes 94
    forced end of volume 104
    freeing temporary storage 113
    journal record synchronization 165
    linking programs 121
    loading a program 123
    multiple events task synchronization 151
    obtain and initialize main storage 117
    obtaining a FWA 79
    program control response codes 128
    random read-only operation 71
    random retrieval for update 73
    random retrieval via indirect access 74
    random update or add data 77
    read variable-length record 102
    release main storage 118
    releasing a FWA 81
    relinquish control to higher priority task 152
    reset sequential retrieval 91
    retrieval of time-ordered data 140
    retrieve temporary data 111
    retrieving selected segments 88
    signal for time expired 135
    single event task synchronization 150
    single-server resource synchronization 153,154
    store temporary data 110
    suspend task processing 133
    synchronous journal output 159

Assembler language (Continued)
  program example (Continued)
    table search using complex table 308
    table search using separate tables 306
    temporary storage response codes 114
    terminal services 66
    terminate sequential retrieval 90
    time of day services 131
    time-ordered task initiation 137
    time service response codes 144
    transfer of program control 122
    transient data response codes 107
    VSAM locate mode I/O 72
    weighted retrieval 330
    write data - predefined symbolic destination 100
  register usage 13
  storage definitions 31
  transfer of control 13
Assembly-time service 17
Asynchronous journal output 161
Asynchronous transaction processing (ATP) 432
ATABLE operand 303
ATI (see automatic task initiation)
ATP (asynchronous transaction processing) 432
ATTACH operand 239
Attaching tasks 146
Attention 181
ATTRB operand 351
Automatic task initiation (ATI) 98

Base addresses 22
BASE operand 343
Basic mapping support (BMS)
  abnormally terminating a logical message 388
  advantages 333
  basic mapping support entry 274
  block data format 334
  data mapping and formatting 334
  disposition and message routing 393
  establishing addressability 360
  examples of use 405
  field data format 334
  implied read/write 338
  input mapping 339
  input operations 361
  input/output mapping 341
  introduction to 333
  message routing 336,388
  non-cumulative page building 378
  offline map building 341
  online map use 360
  output mapping 340
  output operations 364
  page building with mapping 364
  page building without mapping 373
  PAGEBLD overflow processing 370
  paging commands on video devices 405
  physical map 337
  printer control characters 401
Basic mapping support (BMS) (Continued)
  programming considerations 337
  response codes 398
  specifying maps 338
  standard attention identifier list 402
  standard attribute list 401
  status flag byte 400
  symbolic description map 337
  terminal code table 399
  terminal paging 335
  terminating a logical message 384
  test response 394
  text data format 334
  TIOA specification 402
  trace table entry 274
  trailer maps 371
BASIC operand 302
Batch mode application 418
Binary synchronous devices 186
Bit manipulation
  macro instruction 313
  operation 296
  returned values 315
BIT operand 314
BITEST operand 314
BITFLIP operand 314
BITOFF operand 314
BITON operand 314
BITSETOFF operand 314
BITSETON operand 314
BMS (see basic mapping support)
Browsing
  description of 68
  examples 85
  operation 82
  reset 91
  sequential retrieval 83
  skip-sequential processing 82
  terminate 90
Built-in functions
  bit manipulation 296,313
  copying storage referred by BIF 302
  field edit 312
  field verify 296,310
  input formatting 296,315
  listing of 295
  macro instruction 295
  phonetic conversion 295,308
  priority of 295
  requests for 295
  table search 295,303
  trace table entry 276
  weighted retrieval 298,320

CANCEL operand
  DFHIC macro instruction 236
  DFHPC TYPE=ABEND 223
Cancel POST request 141
Card-reader-in-line-printer-out (CRLP) 251
CBUFF operand 182
Chaining of storage areas 23
CHAP operand 240
CHECK operand

CHECK operand (Continued)
DFHBMS TYPE=CHECK 396
DFHFC macro instruction 202
DFHIC macro instruction 237
DFHJC macro instruction 248
DFHPC macro instruction 225
DFHTD macro instruction 209
DFHTS macro instruction 216
CICS operand 289
CICS/VS storage dump 289
CLASS operand 219
COBADDR operand 225
COLUMN operand 348
Common system area (CSA)
addressability of
ANS COBOL 40
assembler language 31
PL/I 51
contents of 24
CWA 25
fields of 24
storage definition
ANS COBOL 40
assembler language 31
PL/I 51
Common work area (CWA)
addressability of
ANS COBOL 40
assembler language 31
PL/I 51
shared 25
size 25
storage definition
ANS COBOL 40
assembler language 31
PL/I 51
uses of 25
COMPLETE operand 290
COND operand
DFHJC TYPE=WRITE 246
DFHPC TYPE=LINK 221
DFHPC TYPE=LOAD 222
DFHSC TYPE=GETMAIN 218
DFHTS TYPE=PUT 211
DFHTS TYPE=PUTQ 212
Conversational mode 66
CONVERSE operand 171
Convert label to address 127
Copy control character (CCC) 183
Copy function 183
COPY operand 183
Copying storage definitions (see storage definitions)
CRLP
(card-reader-in-line-printer-out) 251
CSA (see common system area)
CSACTODB 24
CSAJYDP 24
CSAOPFLA 24
CSATODP 24
CSAWABA 24
CTLCHAR operand 184
CTRL operand
DFHBMS TYPE=OUT 382
DFHBMS TYPE=PAGEBLD 368
DFHBMS TYPE=PAGEOUT 384
CTRL operand (Continued)
DFHBMS TYPE=RETURN 382
DFHBMS TYPE=STORE 382
DFHBMS TYPE=TEXTBLD 376
CURSOR operand
DFHBMS TYPE=OUT 383
DFHBMS TYPE=PAGEBLD 369
DFHBMS TYPE=RETURN 383
DFHBMS TYPE=STORE 383
DFHBMS TYPE=TEXTBLD 377
Customer information control system/virtual storage (CICS/VS)
assembly-time service 17
built-in functions 295
data base concept 3
dump services 287
execution mode 5
introduction 1
macro instruction summary 497
macro instructions 10
major components 3
major responsibilities 2
online environments 8
sample programs 469
sequential terminal support 251
storage areas 481
system management functions 4
system monitoring component 251
transaction flow 6
transaction processing 2
virtual storage environment 15
CWA (see common work area)

DAM data set
adding records 446
browse operation 197
random retrieval 188
retrieval method 188,197
updating nonkeyed 446
Data base concept 3
Data base considerations 432
Data base/data communication (DB/DC) 1
Data Language/I (DL/I)
acquiring an I/O work area 453
building setment search arguments 452
call statement 451,456,458
DL/I requests 462,464,466
issuing the DL/I call 454
macro instruction 450
program communication blocks (PCB) 450
quasi-reentrant considerations 449
releasing a PSB 457
requesting services of 449
requirements for 68
response codes 460
services 272
test response 459
trace table entry 272
Data mapping and formatting (BMS) 334
DATA operand
DFHBMS TYPE=OUT 381
DFHBMS TYPE=PAGEBLD 367
DFHBMS TYPE=RETURN 381
DFHBMS TYPE=STORE 381

DATA operand (Continued)
  DFHMDI macro instruction 346,349
Data set name
  DATASET operand 187
  TCA field 28
DATAID operand
  DFHTS TYPE=GET 213
  DFHTS TYPE=GETQ 214
  DFHTS TYPE=PURGE 216
  DFHTS TYPE=PUT 210
  DFHTS TYPE=PUTQ 211
  DFHTS TYPE=RELEASE 215
  restriction 210
DATASET operand
  DFHBIF TYPE=WTRETST 321
  DFHFC TYPE=DELETE 193
  DFHFC TYPE=GET 187
  DFHFC TYPE=GETAREA 194
  DFHFC TYPE=SETL 196
DATA1 operand 258
DATA1TP operand 258
DATA2 operand 258
DATA2TP operand 258
DCI operand 240
DCT (see destination control table)
DEEDIT operand 312
Deferred journal output 161
DEFLDNM operand 316
Delay task 132
Delete a program 124
DELETE operand
  DFHFC macro instruction 193
  DFHPC macro instruction 223
DEQ operand 241
DESTID operand
  DFHTD TYPE=FEOV 208
  DFHTD TYPE=GET 207
  DFHTD TYPE=PURGE 208
  DFHTD TYPE=PUT 206
Destination control table 26
Destination control table (DCT) 97,99
DFHBFTCA macro instruction
  general format 302
  operands 302
  operation of 302
DFHBIF macro instruction
  prerequisites 295
  priority of 295
  TYPE=BITSETON
    general format 313
    operands 313
    operation of 313
    returned values 315
  TYPE=DEEDIT
    general format 312
    operands 312
    operation of 312
    returned value 313
  TYPE=DEFLDNM
    general format 316
    operands 316
    operation of 316
    required delimiters 317
  TYPE=FVERIFY
    general format 310
    operands 310
DFHBIF macro instruction (Continued)
  TYPE=FVERIFY (Continued)
    operation of 310
    returned values 311
  TYPE=INFORMAT
    error condition 319
    general format 317
    operands 317
    operation of 317
    returned values 319
  TYPE=PHONETIC
    general format 308
    operands 308
    operation of 308
    phonetic coding method 309
    returned value 309
  TYPE=TSEARCH
    complex table 308
    general format 303
    operands 303
    operation of 303
    returned values 306
    separate tables 306
  TYPE=WTRETCHK
    general format 329
    operands 329
    operation of 329
  TYPE=WTRETGET
    general format 327
    operands of 327
    operation of 327
    returned values 327
  TYPE=WTRETREL
    general format 328
    operands 328
    operation of 328
  TYPE=WTRETST
    general format 320
    operands 320
    operation of 320
    returned values 323
  TYPE=WTRTPARM
    general format 323
    operands 323
    operation of 323
DFHBMS macro instruction 360
  TYPE=CHECK
    general format 395
    operands 395
    operation of 395
  TYPE=IN
    general format 361
    operands 361
    operation of 361
  TYPE=MAP
    general format 361
    operands 361
    operation of 361
  TYPE=OUT
    general format 379
    operands 379
    operation of 379
  TYPE=PAGEBLD
    general format 364
    operands 364
    operation of 363

DFHBMS macro instruction (Continued)
TYPE=PAGEBLD (Continued)
programming notes 369
TYPE=PAGEOUT
general format 384
operands 384
operation of 384
programming notes 386
TYPE=PURGE
general format 388
operation of 388
TYPE=RETURN
general format 379
operands 379
operation of 379
TYPE=ROUTE
general format 389
operands 389
operation of 388
programming notes 391
TYPE=STORE
general format 379
operands 379
operation of 379
TYPE=TEXTBLD
general format 373
operands 373
operation of 373
programming notes 377
DFHCOVER macro instruction
placement of 17
use of 17
DFHDC macro instruction
listing of services 287
operation of 287
requirements 287
TYPE=CICS
format of 289
operands 289
operation of 289
TYPE=COMPLETE
format of 290
operands 289
operation of 289
TYPE=PARTIAL
format of 291
operands 291
operation of 291
TYPE=TRANSACTION
format of 288
operands 288
operation of 288
DFHFC macro instruction
file control request/response 28
segmented records 188
TYPE=(DL/I [,function])
general format 454
operands 454
operation of 454
programming notes 456
TYPE=(DL/I,PCB)
general format 450
operands 450
operation of 450
TYPE=(DL/I,T)
general format 457
DFHFC macro instruction (Continued)
TYPE=(DL/I,T) (Continued)
operation of 457
TYPE=CHECK
DL/1 services 461
general format 202,461
operands 202,461
operation of 93,459
response codes 95,460
TYPE=DELETE
general format 192
operands 192
operation of 78
TYPE=ESETL
general format 200
operands 200
operation of 90
TYPE=GET
CICS/VS services for 70
data set name 187
general format 187
index identification 188
operands 187
operation of 69
prerequisites 69
record identification 187
segment set identification 188
TYPE=GETAREA
CICS/VS services for 79
general format 194
operands 194
operation of 78
prerequisites 79
TYPE=GETNEXT
CICS/VS services for 87
general format 198
operands 200
operation of 87
prerequisites 86
TYPE=PUT
CICS/VS services for 76
general format 190
operands 190
operation of 76
prerequisites 76
TYPE=RELEASE
CICS/VS services for 81
general format 195
operands 195
operation of 80
prerequisites 81
TYPE=RESETL
general format 200
operands 200
operation of 91
TYPE=SETL
CICS/VS services for 84
general format 196
operands 196
operation of 83
prerequisites 84
DFHIC macro instruction
interval control request/response 28
listing of services 130
TYPE=CANCEL
general format 236

DFHIC macro instruction (Continued)
TYPE=CANCEL (Continued)
operands 236
operation of 141
TYPE=CHECK
general format 237
operands 237
operation of 142
response codes 147
TYPE=GET
general format 235
operands 235
operation of 139
requirements for 140
TYPE=GETIME
general format 226
operands 226
operation of 131
TYPE=INITIATE
general format 231
operands 231
operation of 136
requirements for 136
TYPE=POST
general format 229
operands 230
operation of 134
requirements for 134
TYPE=PUT
general format 233
operands 233
operation of 138
TYPE=RETRY
general format 236
operands 237
operation of 142
TYPE=WAIT
general format 226
operands 226
operation of 132
DFHJC macro instruction
TYPE=CHECK
general format 248
operands 248
operation of 167
response codes 168
TYPE=GETJCA
general format 243
operands 244
operation of 157
TYPE=PUT
general format 244
operands 244
operation of 159
TYPE=WAIT
general format 247
operands 247
operation of 165
TYPE=WRITE
general format 246
operands 246
operation of 161
DFHKC macro instruction
ILLOGICAL 205
listing of services 145
TYPE=ATTACH
caution of use 146
general format 239
operands 239
operation 145
requirements for 146
TYPE=CHAP
general format 239
operands 239
operation of 149
TYPE=DEQ
general format 241
operands 241
operation of 152
requirements for 153
TYPE=ENQ
general format 240
operands 241
operation of 152
TYPE=NOPURGE
general format 242
operands 242
operation of 155
TYPE=PURGE
general format 242
operands 242
operation of 155
TYPE=WAIT
general format 240
operands 240
operation of 150,151,152
requirements for 150
DFHMDF macro instruction
general format 351
operands 351
operation of 351
programming notes 354
DFHMDI macro instruction
general format 346
operands 346
operation of 346
programming notes 350
DFHMSG macro instruction
general format 341
operands 341
programming notes 345
DFHPC macro instruction
listing of services 119
TYPE=ABEND
general format 223
operands 223
operation of 125
termination code 26
TYPE=CHECK
general format 225
operands 225
operation of 127
requirements for 128
response codes 128
TYPE=COBADDR
general format 225
operands 225
operation of 127
TYPE=DELETE
general format 223

DFHPC macro instruction (Continued)
TYPE=DELETE (Continued)
operands 223
operation of 124
requirements for 124
TYPE=LINK
general format 220
operand 220
operation of 120
requirements for 121
TYPE=LOAD
general format 222
operands 222
operation of 122
prerequisites 122
TYPE=RESETXIT
general format 224
operands 224
operation of 126
TYPE=RETURN
general format 223
operands 223
operation of 123
requirements for 123
TYPE=SETXIT
general format 223
operands 224
operation of 126
prerequisites 126
TYPE=XCTL
general format 220
operation of 121
requirements for 121
DFHSC macro instruction
TYPE=FREEMAIN
general format 220
operands 220
operation of 118
prerequisites 118
TYPE=GETMAIN
general format 218
number of bytes 27
operands 218
operation of 116
prerequisites 117
requirements for 117
storage address returned 26
DFHSP macro instruction
format 250
operand 249
operation of 169
DFHTC macro instruction
address of a TIOA 64
addressability of 60
conversational mode 66
data length 62
dump requirements 63
examples of 66,67
general format 171
input operations 171
list of services 60,61
operands 172
order of operands 61
output operations 177
prerequisites of 62,63
program testing and debugging 252
DFHTC macro instruction (Continued)
saving a TIOA 65
synchronizing terminal input/output 66
TCAM-supported terminals 171
uses of 63,64,65
VTAM-supported logical units 172
DFHTD macro instruction
listing of services 98
transient data control
request/response 28
TYPE=CHECK
general format 209
operands 209
operation of 105
response codes 106
TYPE=FEOV
general format 207
operands 207
operation of 104
TYPE=GET
general format 207
operands 207
operation of 101
use of 102
TYPE=PURGE
general format 208
operands 208
operation of 105
TYPE=PUT
general format 206
operands 206
operation of 99
requirements of 99
use of 100
DFHTR macro instruction
listing of services 254
TYPE=ENTRY
format of 257
operands 257
TYPE=OFF
format of 256
operands 256
TYPE=ON
format of 254
operands 254
operation of 254
DFHTS macro instruction
listing of services 109
temporary storage control
request/response 28
TYPE=CHECK
general format 216
operands 216
operation of 113
response codes 114
TYPE=GET
general format 213
operands 213
operation of 111
TYPE=GETQ
general format 214
operands 214
operation of 111
TYPE=PURGE
general format 216
operands 216

DFHTS macro instruction (Continued)
TYPE=PURGE (Continued)
operation of 112
TYPE=PUT
general format 210
operands 210
operation of 109
TYPE=PUTQ
general format 211
operands 211
operation of 109
TYPE=RELEASE
general format 215
operands 215
operation of 112
DISCONNECT operand 186
DL/I (see Data Language/I)
DL/I operand
function parameter 454
PCB parameter 450
T parameter 457
DL/I,PCB operand 450
DLINA operand 451,461
DMPCODE operand
DFHDC TYPE=CICS 289
DFHDC TYPE=COMPLETE 290
DFHDC TYPE=PARTIAL 291
DFHDC TYPE=TRANSACTION 288
DSECT operand 342
DSIDER operand
DFHBIF TYPE=WTRETCHK 329
DFHBIF TYPE=WTRETST 321
DFHFC TYPE=CHECK 203
DFHFC TYPE=DELETE 194
DFHFC TYPE=GET 190
DFHFC TYPE=GETAREA 195
DFHFC TYPE=SETL 198
Dump services 287
CICS/VS storage dump 289
dump control 287
dump management 287
introduction to 287
macro instruction 287
operation of 287
partial storage dump 291
requests for 287
trace table entry 267
transaction and CICS/VS storage dump 289
transaction storage dump 288
DUPDS operand
DFHFC TYPE=CHECK 204
DFHFC TYPE=GET 190
DUPREC operand
DFHFC TYPE=CHECK 203
DFHFC TYPE=PUT 191

ECADDR operand 240
ENDDATA operand
DFHIC TYPE=CHECK 238
DFHIC TYPE=GET 235
ENDFILE operand
DFHBIF TYPE=WTRETCHK 329
DFHBIF TYPE=WTRETGET 327
ENDFILE operand (Continued)
DFHFC TYPE=CHECK 205
DFHFC TYPE=GETNEXT 200
ENERROR operand
DFHTS TYPE=CHECK 217
DFHTS TYPE=GETQ 215
ENQ operand 241
ENTRY operand
DFHTR macro instruction 257
DFHTS TYPE=GETQ 215
EOB 60
EODPURG operand 386
EOF operand 175
ERASE operand
DFHBMS TYPE=OUT 381
DFHBMS TYPE=PAGEBLD 367
DFHBMS TYPE=RETURN 381
DFHBMS TYPE=STORE 381
DFHBMS TYPE=TEXTBLD 374
DFHTC macro instruction 183
ERASEAUP operand 183
ERROR operand
DFHBIF TYPE=INFORMAT 317
DFHBIF TYPE=PHONETIC 308
DFHBIF TYPE=TSEARCH 308
DFHBMS TYPE=CHECK 397
DFHBMS TYPE=IN 364
DFHBMS TYPE=MAP 364
DFHBMS TYPE=OUT 383
DFHBMS TYPE=PAGEBLD 369
DFHBMS TYPE=PAGEOUT 386
DFHBMS TYPE=RETURN 383
DFHBMS TYPE=ROUTE 391
DFHBMS TYPE=STORE 383
DFHBMS TYPE=TEXTBLD 377
DFHFC TYPE=CHECK 203
DFHFC TYPE=DELETE 194
DFHFC TYPE=ESETL 200
DFHFC TYPE=GET 190
DFHFC TYPE=GETAREA 195
DFHFC TYPE=GETNEXT 200
DFHFC TYPE=PUT 191
DFHFC TYPE=RELEASE 195
DFHFC TYPE=RESETL 202
DFHFC TYPE=SETL 198
DFHIC TYPE=CANCEL 236
DFHIC TYPE=CHECK 238
DFHIC TYPE=GET 235
DFHIC TYPE=GETIME 226
DFHIC TYPE=INITIATE 232
DFHIC TYPE=POST 231
DFHIC TYPE=PUT 235
DFHIC TYPE=RETRY 237
DFHIC TYPE=WAIT 229
DFHTS TYPE=CHECK 217
DFHTS TYPE=GET 214
DFHTS TYPE=GETQ 215
DFHTS TYPE=PURGE 216
DFHTS TYPE=PUT 211
DFHTS TYPE=PUTQ 212
DFHTS TYPE=RELEASE 215
ERRTERM operand 391
ESETL operand 200
Examples of programs
abnormally terminate a transaction 125
acquire the journal control area 157

Examples of programs (Continued)
asynchronous journal output 162
automatic task initiation 138
BMS examples 405
BMS map definition 355
BMS response codes 398
BMS TIOA specification 403
browse operation 85
building a new record 79
change priority of a task 148
copying storage definitions 37,47,57
copying storage referred by BIF 302
delete a program 124
DL/I requests 462,464,466
establish program exit 126
examine TCTTETAB 426
executable CICS/VS sample programs 469
file control response codes 94
forced end of volume 104
freeing temporary storage 113
journal record synchronization 165
linking programs 121
loading a program 123
multiple events task synchronization 151
obtain and initialize main storage 117
obtaining a FWA 79
program control response codes 128
random read-only operation 71
random retrieval for update 73
random retrieval via indirect access 74
random update or add data 77
read variable-length record 102
release main storage 118
releasing a FWA 81
relinquish control to higher priority task 152
reset sequential retrieval 91
retrieval of time-ordered data 140
retrieve temporary data 111
retrieving selected segments 88
signal for time expired 135
single event task synchronization 150
single-server resource synchronization 153,154
store temporary data 110
suspend task processing 133
synchronous journal output 159
table search using complex table 308
table search using separate tables 306
temporary storage response codes 114
terminal services 66
terminate sequential retrieval 90
time of day services 131
time-ordered task initiation 137,138
time service response codes 144
transfer of program control 122
transient data response codes 107
VSAM locate mode I/O 72
weighted retrieval 330
write data - predefined symbolic destination 100
Executable CICS/VS sample programs 469
Expiration of specified time 134
EXPIRD operand
DFHIC TYPE=CHECK 238
EXPIRD operand (Continued)
DFHIC TYPE=POST 231
DFHIC TYPE=WAIT 229
Extrapartition
alignment requirements 104
control processing 104
data 97
indirect destinations 98
queue 97

Facility control area 26
address of 26
FCADDR operand 239
FEOV operand 207
Field edit
macro instruction 312
operation 296
returned value 313
Field engineering (FE) class trace 254
FIELD operand
DFHBIF TYPE=BITSETON 313
DFHBIF TYPE=DEEDIT 312
DFHBIF TYPE=FVERIFY 310
DFHBIF TYPE=PHONETIC 308
Field verify
macro instruction 310
operation 296
returned values 311
FIELDS operand 318
FIELD1 operand 323
FIELD2 operand 324
File control request/response
setting the 28
TCA field 28
File I/O area (FIOA)
address of 28
addressability of
ANS COBOL 42
assembler language 33
PL/I 53
storage definition
ANS COBOL 42
assembler language 33
PL/I 53
use in file services 68
File services
access methods 67
accessing a record 70
browsing 82
data work areas 68
delete data 79
file control 68
file management 67
introduction to 67
macro instruction 187
mass insert 81
obtain a file work area 79
priority of 69
random retrieval 69
read-only retrieval 71
release file storage 80
reset sequential retrieval 91
response codes 95
retrieval for update 73

File services (Continued)
  retrieval via indirect access 74
  retrieve next sequential record 86
  sequential retrieval 83
  summary of 4
  terminate sequential retrieval 90
  test response 93
  trace table entry 268
  update or add data 76
File work area (FWA)
  address of 28
  addressability of
    ANS COBOL 42
    assembler language 34
    PL/I 54
  obtain a file work area 79
  release file storage 80
  storage definition
    ANS COBOL 42
    assembler language 34
    PL/I 54
  use in file services 68
FINAL operand 342
FIOA (see file I/O area)
FORM operand 227
FREEMAIN operand 220
FTABLE operand 304
FUNCNS operand 461
FVERIFY operand 311
FWA (see file work area)

GET operand
  DFHIC macro instruction 235
  DFHTC macro instruction 171,187
  DFHTD macro instruction 207
  DFHTS macro instruction 213
GETAREA operand 194
GETIME operand 226
GETJCA operand 244
GETMAIN operand 218
GETNEXT operand 200
GETQ operand 214
GRPNAME operand 353

HEADER operand
  DFHBMS TYPE=TEXTBLD 374
  DFHMDI macro instruction 349

ICDADDR operand
  DFHIC TYPE=GET 235
  DFHIC TYPE=PUT 234
ID operand 257
IDERROR operand
  DFHJC TYPE=CHECK 248
  DFHJC TYPE=PUT 245
  DFHJC TYPE=WAIT 248
  DFHJC TYPE=WRITE 247
  DFHTD TYPE=CHECK 209
  DFHTD TYPE=FEOV 208
  DFHTD TYPE=PURGE 208
IDERROR operand (Continued)
  DFHTD TYPE=PUT 206
  DFHTS TYPE=CHECK 217
  DFHTS TYPE=GET 214
  DFHTS TYPE=GETQ 215
  DFHTS TYPE=PURGE 216
  DFHTS TYPE=RELEASE 215
ILLOGIC operand
  DFHBIF TYPE=WTRETCHK 329
  DFHBIF TYPE=WTRETGET 327
  DFHBIF TYPE=WTRETREL 328
  DFHBIF TYPE=WTRETST 321
  DFHFC TYPE=CHECK 205
  DFHFC TYPE=DELETE 194
  DFHFC TYPE=ESETL 200
  DFHFC TYPE=GET 190
  DFHFC TYPE=GETNEXT 200
  DFHFC TYPE=PUT 191
  DFHFC TYPE=RELEASE 195
  DFHFC TYPE=RESETL 202
  DFHFC TYPE=SETL 198
IN operand 361
Index identification 168
  INDEX operand 188
  setting the 28
  TCA field 28
INDEX operand
  DFHBIF TYPE=TSEARCH 305
  DFHFC TYPE=GET
Indirect accessing
  duplicate records 441
  general rules 439
  index hierarchy considerations 440
  information required 440
  programming considerations 439
INFORMAT operand 318
INITIAL operand 353
INITIATE operand 231
INITIMG operand
  DFHFC TYPE=GETAREA 194
  DFHSC TYPE=GETMAIN 218
Input formatting
  combination input 298
  fixed format
    error condition 319
    macro instruction 317
    operation 296
    returned values 319
  keyword format
    macro instruction 316
    operation 297
    required delimiters 317
  positional format 297
  storage definition 316
  weighted retrieval
    initiate 320
    operation 298
    release storage areas
    response codes
    returned values 323,327
    selection criteria 323
    test response 328
Input mapping (BMS) 339
Input/output mapping (BMS) 341
INPUTNO operand 321
INPUTPC operand 322

INPUTST operand 322
Inquiry mode application 419
Interval control request/response
  addressability of
    ANS COBOL 44
    assembler language 36
    PL/I 56
  setting the 28
  storage definition
    ANS COBOL 44
    assembler language 36
    PL/I 56
  TCA field 28
Intrapartition
  data 97
  indirect destinations 98
  purge 105
  queue 97
INTRVAL operand
  DFHBMS TYPE=ROUTE 389
  DFHIC TYPE=GETIME 226
  DFHIC TYPE=INITIATE 231
  DFHIC TYPE=POST 230
  DFHIC TYPE=PUT 233
Invalid LDC mnemonic 401
INVET operand
  DFHBMS TYPE=CHECK 396
  DFHBMS TYPE=ROUTE 391
INVMPSZ operand
  DFHBMS TYPE=CHECK 397
  DFHBMS TYPE=IN 364
  DFHBMS TYPE=MAP 364
  DFHBMS TYPE=OUT 383
  DFHBMS TYPE=PAGEBLD 369
  DFHBMS TYPE=RETURN 383
  DFHBMS TYPE=STORE 383
Invoking services 59
INVREQ operand
  DFHBIF TYPE=WTRETCHK 329
  DFHBIF TYPE=WTRETGET 327
  DFHBIF TYPE=WTRETREL 328
  DFHBIF TYPE=WTRETST 321
  DFHBMS TYPE=CHECK 396
  DFHBMS TYPE=OUT 383
  DFHBMS TYPE=PAGEBLD 369
  DFHBMS TYPE=RETURN 383
  DFHBMS TYPE=STORE 383
  DFHBMS TYPE=TEXTBLD 377
  DFHFC TYPE=(DL/I [,function]) 456
  DFHFC TYPE=(DL/I,PCB) 451
  DFHFC TYPE=CHECK 203,462
  DFHFC TYPE=DELETE 194
  DFHFC TYPE=ESETL 200
  DFHFC TYPE=GET 190
  DFHFC TYPE=GETAREA 195
  DFHFC TYPE=GETNEXT 200
  DFHFC TYPE=PUT 191
  DFHFC TYPE=RELEASE 195
  DFHFC TYPE=RESETL 202
  DFHFC TYPE=SETL 198
  DFHIC TYPE=CANCEL 236
  DFHIC TYPE=CHECK 238
  DFHIC TYPE=GET 235
  DFHIC TYPE=GETIME 226
  DFHIC TYPE=INITIATE 232
  DFHIC TYPE=POST 231
INVREQ operand (Continued)
  DFHIC TYPE=PUT 235
  DFHIC TYPE=RETRY 237
  DFHIC TYPE=WAIT 229
  DFHJC TYPE=CHECK 249
  DFHJC TYPE=PUT 245
  DFHJC TYPE=WAIT 248
  DFHJC TYPE=WRITE 247
  DFHTS TYPE=CHECK 217
  DFHTS TYPE=GET 214
  DFHTS TYPE=GETQ 215
  DFHTS TYPE=PURGE 216
  DFHTS TYPE=PUT 211
  DFHTS TYPE=PUTQ 212
  DFHTS TYPE=RELEASE 215
IOERROR operand
  DFHBIF TYPE=WTRETCHK 329
  DFHBIF TYPE=WTRETGET 327
  DFHBIF TYPE=WTRETST 321
  DFHFC TYPE=CHECK 203
  DFHFC TYPE=DELETE 194
  DFHFC TYPE=GET 190
  DFHFC TYPE=GETNEXT 200
  DFHFC TYPE=PUT 191
  DFHFC TYPE=RELEASE 195
  DFHFC TYPE=RESETL 202
  DFHFC TYPE=SETL 198
  DFHIC TYPE=CHECK 238
  DFHIC TYPE=GET 235
  DFHIC TYPE=PUT 235
  DFHIC TYPE=RETRY 237
  DFHJC TYPE=PUT 245
  DFHJC TYPE=WAIT 248
  DFHTD TYPE=CHECK 209
  DFHTS TYPE=CHECK 217
  DFHTS TYPE=GET 214
  DFHTS TYPE=GETQ 215
  DFHTS TYPE=PUT 211
  DFHTS TYPE=PUTQ 212
IOTYPE operand 366,375,380

JCA (see journal control area)
JCDADDR operand
  DFHJC TYPE=PUT 244
  DFHJC TYPE=WRITE 246
JCDLGTH operand
  DFHJC TYPE=PUT 244
  DFHJC TYPE=WRITE 246
JCP (journal control program) 156
JCT (journal control table) 156
JFILEID operand
  DFHJC TYPE=PUT 244
  DFHJC TYPE=WAIT 248
  DFHJC TYPE=WRITE 246
Journal control area (JCA)
  addressability of
    ANS COBOL 45
    assembler language 37
    PL/I 57
  storage definition
    ANS COBOL 45
    assembler language 37
    PL/I 56
Journal control program (JCP) 156

Journal control table (JCT) 156
Journal record 245
Journal services
  acquire the journal control area 157
  asynchronous journal output 161
  create a journal record 159,161
  deferred journal output 161
  introduction to 156
  journal management 156
  journal record 156
  journal record synchronization 165
  macro instruction 243
  priority of 156
  requests for 156
  response codes 168
  summary of 5
  synchronous journal output 159
  test response 167
  trace table entry 273
JTYPEID operand
  DFHJC TYPE=PUT 244
  DFHJC TYPE=WRITE 246
JUSTIFY operand
  DFHBMS TYPE=TEXTBLD 376
  DFHMDF macro instruction 353
  DFHMDI macro instruction 348

LABEL operand
  DFHBIF TYPE=DEFLDNM 316
  DFHPC TYPE=COBADDR 225
LANG operand 343
LANGCON operand 462
LDC operand 366,375,381
LENGTH operand
  DFHBIF TYPE=DEEDIT 312
  DFHBIF TYPE=FVERIFY 311
  DFHBIF TYPE=INFORMAT 317
  DFHMDF macro instruction 352
LERROR operand
  DFHJC TYPE=CHECK 249
  DFHJC TYPE=PUT 245
  DFHJC TYPE=WRITE 247
LINE operand 347
LINEADR operand 184
Link-editing 15
LINK operand 220
Linking programs 121
LIST operand
  DFHBMS TYPE=ROUTE 389
  DFHDC TYPE=PARTIAL 291
LLbb field 214,235
LOAD operand 222
Loading a program 123
LOADLST operand 222
Locate mode 70
Logical unit of work (LUW) 169
LUW (logical unit of work) 169

Macro instructions
  DFHBFTCA macro instruction 302
  DFHBIF macro instructions 302
  DFHBMS macro instruction 361
Macro instructions (Continued)
  DFHDC macro instruction 287
  DFHFC macro
   instruction 190,203,450,454,457,461
  DFHIC macro instruction 130,226
  DFHJC macro instruction 157,243,246
  DFHKC macro instruction 145,239
  DFHMDF macro instruction 350
  DFHMDI macro instruction 346
  DFHMSD macro instruction 341
  DFHPC macro instruction 119,220
  DFHSC macro instruction 116,218
  DFHSP macro instruction 169,250
  DFHTC macro instruction 61,171,187
  DFHTD macro instruction 98,206
  DFHTR macro instruction 254
  DFHTS macro instruction 109,210
  general format 10
  name field restriction 11
  operand field 11
  operation field 10
  summary of 497
MAP operand
  DFHBMS TYPE=IN 361,361,363
  DFHBMS TYPE=MAP 363
  DFHBMS TYPE=OUT 381
  DFHBMS TYPE=PAGEBLD 367
  DFHBMS TYPE=RETURN 381
  DFHBMS TYPE=STORE 381
  DFHMSD macro instruction 342
MAPADR operand
  DFHBMS TYPE=IN 363
  DFHBMS TYPE=MAP 362
  DFHBMS TYPE=OUT 382
  DFHBMS TYPE=RETURN 382
  DFHBMS TYPE=STORE 382
MAPFAIL operand
  DFHBMS TYPE=CHECK 396
  DFHBMS TYPE=IN 364
  DFHBMS TYPE=MAP 364
MAPSET operand
  DFHBMS TYPE=IN 363
  DFHBMS TYPE=MAP 363
  DFHBMS TYPE=OUT 382
  DFHBMS TYPE=PAGEBLD 368
  DFHBMS TYPE=RETURN 382
  DFHBMS TYPE=STORE 382
Mass insert 81
MATCH operand 325
Message routing
  description of 388
  disposition and message routing 393
  macro instruction 388
  programming notes 391
  status flag byte 400
Mnemonics 485
MODE operand
  DFHFC TYPE=GET 189
  DFHFC TYPE=SETL 198
  DFHMSD TYPE=DSECT 342
Move mode 70
MSETADR operand
  DFHBMS TYPE=IN 363
  DFHBMS TYPE=MAP 363
  DFHBMS TYPE=OUT 382
  DFHBMS TYPE=PAGEBLD 368

MSETADR operand (Continued)
DFHBMS TYPE=RETURN 382
DFHBMS TYPE=STORE 382
Multiple events task synchronization 151
Multithreading 7

NAMES operand
DFHBIF TYPE=DEFLDNM 316
DFHBIF TYPE=INFORMAT 317
NOEDIT operand 381
NOMATCH operand
DFHBIF TYPE=TSEARCH 303
DFHBIF TYPE=WTRTPARM 325
NOPURGE operand 242
NORESP operand
DFHBIF TYPE=WTRETCHK 329
DFHBIF TYPE=WTRETGET 327
DFHBIF TYPE=WTRETREL 328
DFHBIF TYPE=WTRETST 321
DFHBMS TYPE=CHECK 396
DFHBMS TYPE=IN 364
DFHBMS TYPE=MAP 364
DFHBMS TYPE=OUT 383
DFHBMS TYPE=PAGEBLD 369
DFHBMS TYPE=PAGEOUT 386
DFHBMS TYPE=RETURN 383
DFHBMS TYPE=ROUTE 391
DFHBMS TYPE=STORE 383
DFHBMS TYPE=TEXTBLD 377
DFHFC TYPE=(DL/I [,function]) 456
DFHFC TYPE=(DL/I,PCB) 451
DFHFC TYPE=CHECK 202,461
DFHFC TYPE=DELETE 194
DFHFC TYPE=ESETL 200
DFHFC TYPE=GET 190
DFHFC TYPE=GETAREA 195
DFHFC TYPE=GETNEXT 200
DFHFC TYPE=PUT 191
DFHFC TYPE=RELEASE 195
DFHFC TYPE=RESETL 202
DFHFC TYPE=SETL 198
DFHIC TYPE=CANCEL 236
DFHIC TYPE=CHECK 237
DFHIC TYPE=GET 235
DFHIC TYPE=GETIME 226
DFHIC TYPE=INITIATE 232
DFHIC TYPE=POST 231
DFHIC TYPE=PUT 235
DFHIC TYPE=RETRY 237
DFHIC TYPE=WAIT 229
DFHJC TYPE=CHECK 248
DFHJC TYPE=PUT 245
DFHJC TYPE=WAIT 248
DFHJC TYPE=WRITE 247
DFHPC TYPE=CHECK 225
DFHPC TYPE=LINK 221
DFHPC TYPE=LOAD 222
DFHPC TYPE=SETXIT 224
DFHTD TYPE=CHECK 209
DFHTD TYPE=FEOV 208
DFHTD TYPE=PURGE 208
DFHTD TYPE=PUT 206
DFHTS TYPE=CHECK 217
DFHTS TYPE=GET 214
NORESP operand (Continued)
DFHTS TYPE=GETQ 215
DFHTS TYPE=PURGE 216
DFHTS TYPE=PUT 211
DFHTS TYPE=PUTQ 212
DFHTS TYPE=RELEASE 215
NOSPACE operand
DFHFC TYPE=CHECK 203
DFHTD TYPE=CHECK 209
DFHTD TYPE=PUT 206
DFHTS TYPE=CHECK 216
DFHTS TYPE=PUT 211
DFHTS TYPE=PUTQ 212
NOTFND operand
DFHBIF TYPE=WTRETCHK 329
DFHBIF TYPE=WTRETGET 327
DFHBIF TYPE=WTRETST 321
DFHFC TYPE=CHECK 203
DFHFC TYPE=DELETE 194
DFHFC TYPE=GET 190
DFHFC TYPE=GETNEXT 200
DFHFC TYPE=RESETL 202
DFHFC TYPE=SETL 198
DFHIC TYPE=CANCEL 236
DFHIC TYPE=CHECK 238
DFHIC TYPE=GET 235
DFHIC TYPE=RETRY 237
NOTOPEN operand 462
DFHBIF TYPE=WTRETCHK 329
DFHBIF TYPE=WTRETGET 327
DFHBIF TYPE=WTRETST 321
DFHFC TYPE=(DL/I [,function]) 456
DFHFC TYPE=CHECK 205,461
DFHFC TYPE=DELETE 194
DFHFC TYPE=GET 190
DFHFC TYPE=GETAREA 195
DFHFC TYPE=GETNEXT 200
DFHFC TYPE=PUT 191
DFHFC TYPE=RESETL 202
DFHFC TYPE=SETL 198
DFHJC TYPE=CHECK 249
DFHJC TYPE=PUT 245
DFHJC TYPE=WAIT 248
DFHJC TYPE=WRITE 247
DFHTD TYPE=CHECK 209
DFHTD TYPE=FEOV 208
DFHTD TYPE=PUT 206
NOTRANSLATE operand 185
NRECDS operand 322
NULL operand 325
NUMBYTE operand 218
NUMERIC operand 311

OCCURS operand 354
Offline map building (BMS) 341
OFLOW operand
DFHBIF TYPE=WTRETCHK 329
DFHBIF TYPE=WTRETGET 327
DFHBIF TYPE=WTRETST 321
DFHBMS TYPE=PAGEBLD 369
ON operand 254
Online map use (BMS) 360
OPCLASS operand 390
OPTION operand 302

Optional features list (CSA) 24
ORDER operand 305
OUT operand
  DFHBMS TYPE=OUT 379
  DFHBMS TYPE=PAGEBLD 365
  DFHBMS TYPE=TEXTBLD 374
Output mapping (BMS) 340
Override of uppercase 175

PACKED operand 311
Page building
  abnormally terminating a logical message 388
  message routing 388
  non-cumulative 378
  operation 335
  overflow processing 370,372
  paging commands on video devices 405
  terminating a logical message 384
  trailer maps 371
  with mapping 364
  without mapping 373
PAGE operand 171
Page queuing facility 108
PAGEBLD operand 365
PAGEOUT operand 384
Paging commands on video devices 405
PARTIAL operand 291
Partial storage dump 291
PASSBK operand 182
Passing program control 120
PBSBNA operand 462
PCB (program communication blocks) 450
PCB operand 454
PFXADDR operand
  DFHJC TYPE=PUT 245
  DFHJC TYPE=WRITE 246
PFXLGTH operand
  DFHJC TYPE=PUT 245
  DFHJC TYPE=WRITE 246
PGMIDER operand
  DFHPC TYPE=CHECK 225
  DFHPC TYPE=LINK 221
  DFHPC TYPE=LOAD 222
  DFHPC TYPE=SETXIT 224
Phonetic converstion
  macro instruction 308
  operation 295
  phonetic coding method 309
  returned value 309
  subroutine 309
PHONETIC operand 308
Physical map (BMS) 337
PICIN operand 354
PICOUT operand 354
PL/I
  %INCLUDE 20
  addressability of storage areas 21
  link-editing 15
  program example
    abnormally terminate a transaction 125
    acquire the journal control area 157
    asynchronous journal output 162
PL/I (Continued)
  program example (Continued)
    automatic task initiation 138
    BMS examples 405
    BMS map definition 355
    BMS response codes 398
    BMS TIOA specification 403
    browse operation 85
    building a new record 79
    change priority of a task 148
    copying storage definitions 57
    copying storage referred by BIF 302
    delete a program 124
    DL/I requests 466
    establish program exit 126
    examine TCTTETAB 426
    executable CICS/VS sample program 478
    file control response codes 94
    forced end of volume 104
    freeing temporary storage 113
    journal record synchronization 165
    linking programs 121
    loading a program 123
    multiple events task synchronization 151
    obtain and initialize main storage 117
    obtaining a FWA 79
    program control response codes 128
    random read-only operation 71
    random retrieval for update 73
    random retrieval via indirect access 74
    random update or add data 77
    read variable-length record 102
    release main storage 118
    releasing a FWA 81
    relinquish control to higher priority task 152
    reset sequential retrieval 91
    retrieval of time-ordered data 140
    retrieve temporary data 111
    retrieving selected segments 88
    signal for time expired 135
    single event task synchronization 150
    single-server resource synchronization 153,154
    store temporary data 110
    suspend task processing 133
    synchronous journal output 159
    temporary storage response codes 114
    terminal services 66
    terminate sequential retrieval 90
    time of day services 131
    time-ordered task initiation 137,138
    time service response codes 144
    transfer of program control 122
    transient data response codes 107
    VSAM locate mode I/O 72
    weighted retrieval 330
    write data - predefined symbolic destination 100
  register usage 13
  restrictions 15
  storage definitions 51
  transfer of control 13

POS operand 351
POST operand 230
PPT (processing program table) 26
PRINT operand 183
Print request support 64
Printer control characters (BMS) 401
Priority of a task 148
Processing program table (PPT) 26
Program communication blocks 450
Program initialization 13
PROGRAM operand
  DFHDC TYPE=PARTIAL 291
  DFHPC TYPE=DELETE 223
  DFHPC TYPE=LINK 221
  DFHPC TYPE=LOAD 222
  DFHPC TYPE=SETXIT 224
  DFHPC TYPE=XCTL 222
Program services
  abnormally terminate a transaction 125
  communication and logical relationships 120
  convert label to address 127
  delete a program 124
  introduction to 119
  load a program 122
  logical levels 119
  macro instruction 220
  pass control-anticipating return 120
  program management 119
  response codes 128
  return program control 123
  summary of 4
  test response 127
  trace table entry 263
  transfer control 121
Program testing and debugging
  built-in functions 295
  card-reader-in-line-printer-out (CRLP) 251
  dump management 251
  dump services 287
  input data for 251
  introduction to 251
  sequential terminal support 251
  trace management 251
  trace services 253
Programmable device considerations 416
Programming considerations
  asynchronous transaction processing 432
  data base considerations
    adding records to DAM data sets 446
    duplicate records 441
    indirect accessing 439
    record identification field 444
    segmented records 433
    updating nonkeyed DAM data sets 446
  non-programmable device considerations
    2260/2265 considerations 421
    2741 considerations 430
    2770/2780 considerations 423
    2980 considerations 423
    3270 operating in 2260 compatibility mode 422
    3270 print function 431
    7770 considerations 429
  programmable device considerations
Programming considerations (Continued)
  programmable device considerations (Continued)
    System/7 considerations 419
    3735 considerations 417
  terminal-oriented task identification 415
PRTY operand 240
PSB (program specification blocks) 450
PSB operand 450
PSBFAIL operand 451,461
PSBNF operand 451,461
PSBSCH operand 451,461
PSEUDOBIN operand 173
PURGE operand
  DFHBMS TYPE=PURGE 388
  DFHKC macro instruction 242
  DFHTD macro instruction 208
  DFHTS macro instruction 216
PUT operand
  DFHFC macro instruction 190
  DFHIC macro instruction 233
  DFHTC macro instruction 171
  DFHTD macro instruction 206
  DFHTS macro instruction 210,210
PUTQ operand 211

QARGADR operand
  DFHKC TYPE=DEQ 241
  DFHKC TYPE=ENQ 241
QARGLNG operand
  DFHKC TYPE=DEQ 241
  DFHKC TYPE=ENQ 241
Quasi-reentrance 9
QUEBUSY operand 207
QUEZERO operand
  DFHTD TYPE=CHECK 209
  DFHTD TYPE=GET 207

Random
  data retrieval 69
  delete data 78
  read-only retrieval 71
  retrieval for update 73
  retrieval via indirect access 74
  update or add data 76
RANGE operand
  DFHBIF TYPE=TSEARCH 303
  DFHBIF TYPE=WTRTPARM 325
RDATA1 operand 258
RDATA2 operand 258
RDATT operand
  DFHBMS TYPE=IN 364
  DFHBMS TYPE=MAP 364
  DFHTC macro instruction 174
RDIDADR operand
  DFHBIF TYPE=WTRETST 321
  DFHFC TYPE=DELETE 193
  DFHFC TYPE=GET 187
  DFHFC TYPE=PUT 190
  DFHFC TYPE=SETL 196
Read attention support

Read attention support (Continued)
operation 65
programming considerations 430
READ operand
DFHTC macro instruction 171
saving a TIOA 65
uses of 65
READB operand 174
READL operand 183
Record identification
address of 28,187,191
ARGTYP operand 189
field structure 444
programming considerations 444
RDIDADR operand 187
TCA field 28
Recovery/restart services
macro instruction 250
summary of 5
sync point management 169
Register usage 13
RELEASE operand
DFHFC macro instruction 195
DFHSC TYPE=FREEMAIN 220
DFHTS macro instruction 215
DFHTS TYPE=GET 213
Relinquish control to higher priority task 152
REQID operand
DFHIC TYPE=CANCEL 236
DFHIC TYPE=GETIME 226
DFHIC TYPE=INITIATE 232
DFHIC TYPE=POST 230
DFHIC TYPE=PUT 234
RESET operand 186
RESETL operand 200
RESETXIT operand 224
Response codes
BMS 398
DL/I services 460
file control 95
interval control 143
journal control 168
program control 128
temporary storage control 114
transient data control 106
Restrictions
ANS COBOL 14
link-editing 15
PL/I 15
RETFAIL operand
DFHBMS TYPE=CHECK 396
DFHBMS TYPE=ROUTE 391
RETMETH operand
DFHFC TYPE=GET 188
DFHFC TYPE=SETL 197
RETPAGE operand
DFHBMS TYPE=CHECK 396
DFHBMS TYPE=OUT 383
DFHBMS TYPE=PAGEBLD 369
DFHBMS TYPE=PAGEOUT 386
DFHBMS TYPE=RETURN 383
DFHBMS TYPE=STORE 383
DFHBMS TYPE=TEXTBLD 377
Retrieve time-ordered data 139
RETRY operand 237
RETURN operand
DFHBMS TYPE=PAGEBLD 367
DFHBMS TYPE=RETURN 379
DFHBMS TYPE=TEXTBLD 374
DFHPC TYPE=RETURN 223
Return program control 123
ROUTE operand 389
ROUTINE operand 224
RTESOME operand
DFHBMS TYPE=CHECK 397
DFHBMS TYPE=ROUTE 391

SAA (storage accounting area) 20
Sample programs 469
SAVE operand
DFHBMS TYPE=IN 361
DFHBMS TYPE=MAP 361
DFHBMS TYPE=OUT 381
DFHBMS TYPE=PAGEBLD 367
DFHBMS TYPE=RETURN 381
DFHBMS TYPE=STORE 381
DFHBMS TYPE=TEXTBLD 374
DFHTC macro instruction 171
SEGIDER operand
DFHFC TYPE=CHECK 203
DFHFC TYPE=GET 190
DFHFC TYPE=GETNEXT 200
DFHFC TYPE=RESETL 202
DFHFC TYPE=SETL 198
SEGMENT operand 292
Segment set identification
SEGSET operand 188
setting the 28
TCA field 28
Segmented records
defined 433
formats 434
general rules 433
indicators 435
main storage processing of 437
programming considerations 433
root segment 434
search arguments 452
sequence of 434
sets 438
SEGSET operand
DFHFC TYPE=GET 188
DFHFC TYPE=GETNEXT 200
DFHFC TYPE=PUT 191
DFHFC TYPE=RESETL 201
DFHFC TYPE=SETL 196
Sequential
reset sequential retrieval 91
retrieval 83
retrieve next sequential record 86
skip-sequential processing 82
terminal support 251
terminate sequential retrieval 90
Service invocation
file services 68
journal services 156
listing of services 59
overview 59
program services 119

Service invocation (Continued)
recovery/restart services 169
storage services 116
task services 145
temporary storage services 107
terminal services 60
time services 130
transient data services 97
SETL operand 196
SETXIT operand 224
Single event task synchronization 150
Single-server resource synchronization 152
Single threading 7
SIZE operand 347
Skip-sequential processing 82,87
SRCHTYP operand
DFHFC TYPE=DELETE 193
DFHFC TYPE=GET 189
DFHFC TYPE=RESETL 201
DFHFC TYPE=SETL 197
SSALIST operand 455
SSAS operand 455
Standard attention identifier list (BMS) 402
Standard attribute list (BMS) 401
STARTIO operand 246
Storage accounting area (SAA) 19
Storage areas
addressability of (see addressability)
base addresses 22
chaining 23
definitions (see storage definition)
list of 19
required 24
summary of 481
symbolic names 22
types of 19
Storage definition
addressability 21
base addresses 22
chaining of storage areas 23
common system area (CSA)
ANS COBOL 40
assembler language 31
PL/I 51
common work area (CWA)
ANS COBOL 40
assembler language 31
PL/I 51
file input/output area (FIOA)
ANS COBOL 42
assembler language 33
PL/I 53
file work area (FWA)
ANS COBOL 42
assembler language 34
PL/I 54
journal control area (JCA)
ANS COBOL 44
assembler language 37
PL/I 56
recommendation 13
required storage areas 24
storage accounting area (SAA)
ANS COBOL 44
Storage definition (Continued)
storage accounting area (SAA) (Continued)
assembler language 36
PL/I 56
storage accounting field 20
summary of CICS/VS storage areas 481
task control area (TCA)
ANS COBOL 41
assembler language 32
PL/I 52
temporary storage input/output area (TSIOA)
ANS COBOL 43
assembler language 36
PL/I 55
terminal control table terminal entry (TCTTE)
ANS COBOL 40
assembler language 31
PL/I 52
terminal input/output area (TIOA)
ANS COBOL 41
assembler language 33
PL/I 52
transaction work area (TWA)
ANS COBOL 41
assembler language 32
PL/I 52
transient data input area (TDIA)
ANS COBOL 43
PL/I 54
transient data output area (TDOA)
ANS COBOL 43
assembler language 35
PL/I 55
VSAM work area (VSWA) 34
ANS COBOL 42
assembler language 34
PL/I 54
Storage services
accounting for storage 116
activate ABEND exit 126
cancel ABEND exit 126
introduction to 116
macro instruction 218
obtain and initialize main storage 116
reactivate ABEND exit 126
release main storage 118
storage control 116
storage management 116
summary of 4
trace table entry 262
STORE operand 374
DFHBMS TYPE=PAGEBLD 367
DFHBMS TYPE=STORE 379
DFHBMS TYPE=TEXTBLD 374
STORFAC operand
DFHTS TYPE=PUT 210
DFHTS TYPE=PUTQ 212
STYPE operand
DFHTR TYPE=ENTRY 257
DFHTR TYPE=OFF 256
DFHTR TYPE=ON 254
SUBST operand 305
Summary of CICS/VS storage areas 481
Suspend data set 108

Switched lines disconnect 186
Symbolic description map (BMS) 337
Symbolic storage definitions (see storage definition)
Sync point management
  macro instruction 250
  summary of 5
  sync point 169
  trace table entry 285
Synchronize a task 148,150
Synchronizing terminal input/output 66
Synchronous journal output 159
System management functions
  file services 67
  journal services 156
  macro instructions 171
  program services 119
  recovery/restart services 169
  storage services 116
  summary of 4,5
  task services 144
  temporary storage services 107
  time services 130
  transaction flow 6
  transient data services 97
System monitoring component 251
System/7
  data translation 173
  programming considerations 418
  transparent mode 179

Table search
  complex table 308
  macro instruction 303
  operation 295
  returned values 306
  separate tables 307
TARGET operand 303
Task control area (TCA)
  addressability
    ANS COBOL 41
    assembler language 32
    PL/I 52
  contents of 25
  fields of 26
  logical sections 25
  storage definition
    ANS COBOL 41
    assembler language 32
    PL/I 52
Task services
  attaching tasks 146
  change priority of a task 148
  initiate a task 144
  listing of 145
  macro instruction 239
  multiple events task synchronization 151
  relinquish control to higher priority task 152
  single event task synchronization 150
  single-server resource synchronization 152
  summary of 5
Task services (Continued)
  synchronize a task 150
  task control 144
  task management 144
  task purgeability on system overload 155
  task synchronization 148
  trace table entry 261
Task synchronization 132
TASKNA operand 462
TCA (see task control area)
TCADCNB 27
TCAFCAA 28
TCAFCAAA 26
TCAFCAI 28
TCAFCDI 27
TCAFCNRD 28
TCAFCRI 28
TCAFCSI 28
TCAFCTR 28
TCAFCURL 28
TCAICTR 29
TCAM-supported terminals
  input operations 171
  output operations 177
TCAPCAC 26
TCAPCPI 26
TCASCIB 27
TCASCNB 27
TCASCSA 27
TCATDTR 28
TCATSTR 29
TCTTE (see terminal control table terminal entry)
TDADDR operand 206
TDIA (see transient data input area)
TDOA (see transient data output area)
Temporary storage control request/response
  setting the 28
  TCA field 28
Temporary storage I/O area (TSIOA)
  addressability of 35
    ANS COBOL 44
    assembler language 35
    PL/I 55
  obtaining a 219
  storage definition
    ANS COBOL 44
    assembler language 35
    PL/I 55
Temporary storage services
  free temporary data 112
  introduction to 108
  macro instruction 210
  page queuing facility 108
  response codes 114
  retrieve temporary data 111
  storage temporary data 109
  summary of 4
  temporary storage control 107
  temporary storage management 107
  test response 113
  trace table entry 271
TERM operand 343
Terminal code table 399
Terminal control table terminal entry (TCTTE)

Terminal control table terminal entry (TCTTE) (Continued)
address of 26
addressability of
ANS COBOL 40
assembler language 32
PL/I 52
storage definition
ANS COBOL 40
assembler language 31
PL/I 52
Terminal I/O area (TIOA)
addressability of
after a read 65
ANS COBOL 41
assembler language 33
PL/I 52
BMS 403
obtaining a 219
prefix 62
storage definition
ANS COBOL 41
assembler language 33
PL/I 52
Terminal-oriented task identification 415
Terminal paging
BMS 335,364
temporary storage services 107
Terminal services
access methods 60
compatibility mode 63
conversational mode 66
examples of 66,67
macro instruction 171
operating mode 60
read data from a terminal 65
requests for 60
requirement for sequential devices 60
summary of 4
synchronizing terminal input/output 66
terminal control 60
terminal management 60
write data to a terminal 64
Termination code (ABEND) 26
TERMNS operand 461
Test response
BMS services 394
DL/I services 459
file service 202
file services 69,93
journal services 167
program services 127
temporary storage services 113
time services 142
transient data services 105
weighted retrieval 328,330
TEXT operand
DFHBMS TYPE=IN 361
DFHBMS TYPE=MAP 361
DFHTC macro instruction 175
TEXTBLD operand 373
TIMADR operand 227
Time of day 130
TIME operand
DFHBMS TYPE=ROUTE 390
DFHIC TYPE=GETIME 226
TIME operand (Continued)
DFHIC TYPE=INITIATE 232
DFHIC TYPE=POST 230
DFHIC TYPE=PUT 234
Time services
cancel INITIATE or PUT 142
cancel POST request 141
cancel WAIT request 142
delay task 132
expiration of specified time 134
introduction to 130
listing of 130
macro instruction 226
response codes 143
retrieve time-ordered data 139
retry capability 142
summary of 5
task initiation with data 142
task initiation without data 136
task synchronization 132
test response 142
time adjustment feature 130
time of day format 130
time of day services 131
time-ordered cancellation 141
time-ordered data 139
time-ordered task initiation 136
trace table entry 264
TIOA (see terminal I/O area)
TIOATDL 175
TITLE operand 390
Trace services
function of 253
introduction to 253
macro instruction 254
trace control 253
trace entry format 258
trace entry function 257
trace off function 255
trace ON function 254
trace system symbols 255
trace table
built-in functions 276
CICS/VS-DL/I interface entry 272
dump control entry 267
duplicate entries 260
field engineering (FE) entry 286
file control entry 268
interval control entry 264
journal control entry 273
location of 253
program control entry 263
storage control entry 262
sync point program entry 285
task control entry 261
temporary storage control entry 271
trace control entry 285
trace entry general format 259
trace header 259
transient data control entry 270
VTAM terminal control 278
Trace system symbols 255
TRAILER operand
DFHBMS TYPE=PAGEOUT 385
DFHBMS TYPE=TEXTBLD 376
DFHMDI macro instruction 349

Transaction
  asynchronous processing 432
  flow 6
  processing 2
Transaction and CICS/VS storage dump 289
TRANSACTION operand
  DFHDC TYPE=PARTIAL 291
  DFHDC TYPE=TRANSACTION 288
Transaction storage dump 288
Transaction work area (TWA)
  addressability of
    ANS COBOL 41
    assembler language 32
    PL/I 52
  description of 29
  size of 29
  storage definition
    ANS COBOL 41
    assembler language 32
    PL/I 52
Transfer of control 13,122
TRANSID operand
  DFHBMS TYPE=PAGEOUT 385
  DFHIC TYPE=INITIATE 232
  DFHIC TYPE=PUT 234
  DFHKC TYPE=ATTACH 239
  DFHPC TYPE=RETURN 223
Transient data control request/response
  setting the 28
  TCA field 28
Transient data input area (TDIA)
  addressability of
    ANS COBOL 43
    assembler language 34
    PL/I 54
  obtaining a 219
  storage definition
    ANS COBOL 43
    assembler language 34
    PL/I 54
Transient data output area (TDOA)
  addressability of
    ANS COBOL 43
    assembler language 35
    PL/I 55
  obtaining a 219
  storage definition
    ANS COBOL 43
    assembler language 35
    PL/I 55
Transient data services
  acquire queued data 101
  automatic task initiation (ATI) 98
  dispose of data 99
  extrapartition data 98
  forced end of volume 104
  indirect destinations 98
  intrapartition data 97
  introduction to 97
  macro instruction 206
  purge intrapartition data 105
  response codes 106
  summary of 4
  test response 105
  trace table entry 270
  transient data control 97
Transient data services (Continued)
  transient data management 97
Translate tables for the 2980 523
Transparent mode 179
TRANSPARENT operand 179
TRMIDER operand
  DFHIC TYPE=CHECK 238
  DFHIC TYPE=INITIATE 232
  DFHIC TYPE=PUT 235
TRMIDNT operand
  DFHIC TYPE=INITIATE 232
  DFHIC TYPE=PUT 234
TRNIDER operand
  DFHIC TYPE=CHECK 238
  DFHIC TYPE=INITIATE 232
  DFHIC TYPE=PUT 235
TSDADDR operand
  DFHTS TYPE=GET 213
  DFHTS TYPE=GETQ 214
  DFHTS TYPE=PUT 210
  DFHTS TYPE=PUTQ 212
TSEARCH operand 303
TSINVLD operand
  DFHIC TYPE=CHECK 238
  DFHIC TYPE=GET 235
TSIOA (see temporary storage I/O area)
TSIOERR operand
  DFHBMS TYPE=CHECK 396
  DFHBMS TYPE=OUT 383
  DFHBMS TYPE=PAGEBLD 369
  DFHBMS TYPE=PAGEOUT 386
  DFHBMS TYPE=RETURN 383
  DFHBMS TYPE=STORE 383
  DFHBMS TYPE=TEXTBLD 377
TWA (see transaction work area)
TYPOPER operand
  DFHFC TYPE=GET 188
  DFHFC TYPE=GETAREA 194
  DFHFC TYPE=PUT 191

USER operand 250

Virtual storage
  concepts 15
  techniques 9,15
VSAM data set
  browse operation 197,201
  mass insert 191
  random retrieval 189
  variable-length records 71
VSAM work area (VSWA)
  address of 28
  addressability of
    ANS COBOL 43
    assembler language 34
    PL/I 54
  storage definition
    ANS COBOL 42
    assembler language 34
    PL/I 54
VSWA (see VSAM work area)
VTAM-supported logical units

VTAM-supported logical units (Continued)
input operations 172
output operations 177
VTAM trace table entry 278

WAIT operand
DFHBMS TYPE=OUT 379
DFHBMS TYPE=RETURN 379
DFHBMS TYPE=STORE 379
DFHBMS TYPE=TEXTBLD 374
DFHIC macro instruction 226
DFHJC macro instruction 247
DFHKC macro instruction 240
DFHTC macro instruction 171
uses of 66
Weighted retrieval
initiate 320
macro instruction 320
operation 298
release storage areas 327
response codes 330
rest response 328
retrieve selected records 326
selection criteria 323
selection of records 300
weighted qualification percentage (WQP) 299
WQP (weighted qulification percentage) 299
WRBRK operand
DFHBMS TYPE=OUT 383
DFHBMS TYPE=PAGEBLD 369
DFHBMS TYPE=PAGEOUT 387
DFHBMS TYPE=RETURN 383
DFHBMS TYPE=STORE 383
DFHBMS TYPE=TEXTBLD 377
DFHTC macro instruction 181
Write break support
operation 64
programming considerations 431
WRITE operand
DFHJC macro instruction 246
DFHTC macro instruction 171
uses of 63,64
WRITEL operand 183
WRKAREA operand 454
WTRET operand 302
WTRETCHK operand 329
WTRETGET operand 327
WTRETREL operand 328
WTRETST operand 321
WTRTPARM operand 323

XCTL operand 222

2260 compatibility mode (3270)
erasing the screen 64
input operations 174
output operations 183
print request support 64
2260 compatibility mode (3270) (Continued)
programming considerations 423
2260 display station
input operations 173
output operations 180
2260/2265 terminals 421
2741 communication terminal
attention 181
input operations 173
programming considerations 430
read attention support 65,174
write break support 64,181
2770 data communication system
output operations 179
programming considerations 423
2780 data transmission system
output operations 179
programming considerations 423
2980 general banking terminal system
common buffer message 182
data handling 425
high-level-language programs 425
passbook control 423
printing a passbook 182
programming considerations 182,423
segmented writes control 424

3270 information display system
copy function 183
erase screen 183
erasing the screen 64
input operations 174
locking the keyboard 183
output operations 182
override of uppercase 175
override of uppercase translation 65
print function 431
print request support 64
programming considerations 431
write control character (WCC) 184
2260 compatibility mode 174,174,183,422
3735 programmable buffered terminal
autoanswer 418
autocall 418
end of file 175
EOF condition 175
form description program (FDP) 185
input operations 175
output operations 185
programming considerations 417
3740 considerations 418
3740 data entry system
input operations 176
output operations 185
3780 data communications 179

7770 audio response unit
programming considerations 429
ready message 63

Customer Information
Control System/Virtual
Storage (CICS/VS)
Application Programmer's
Reference Manual

SH20-9003-2

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such request, please contact your IBM representative or the IBM Branch Office serving your locality.*

CUT ALONG DOTTED LINE

Reply requested:
Yes ☐
No ☐

Name: _____
Job Title: _____
Address: _____
_____ Zip _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

fold fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 813 HP
1133 Westchester Avenue
White Plains, New York 10604

fold fold

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)

CICS/VS Application Programmer's Reference Manual Printed in U.S.A. SH20-9003-2

SH20-9003-2

CICS/VS Application Programmer's Reference Manual Printed in U.S.A. SH20-9003-2

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)